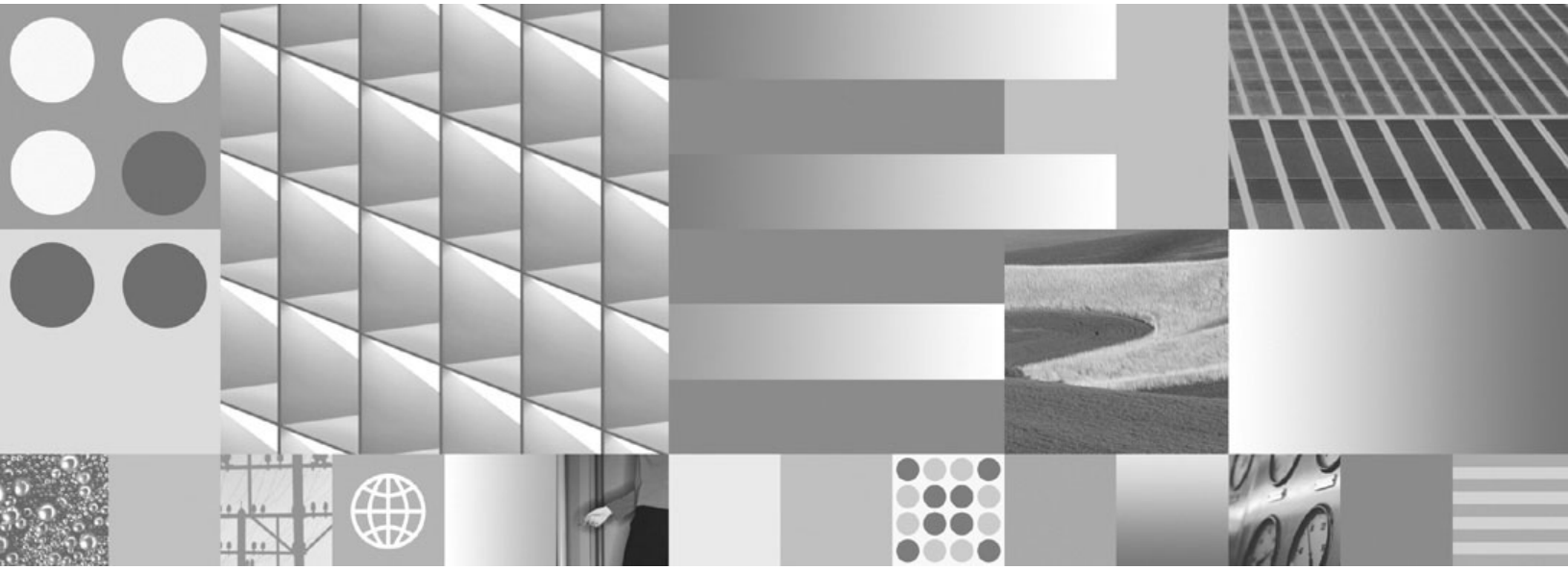
IBM Informix

Verze 11.1

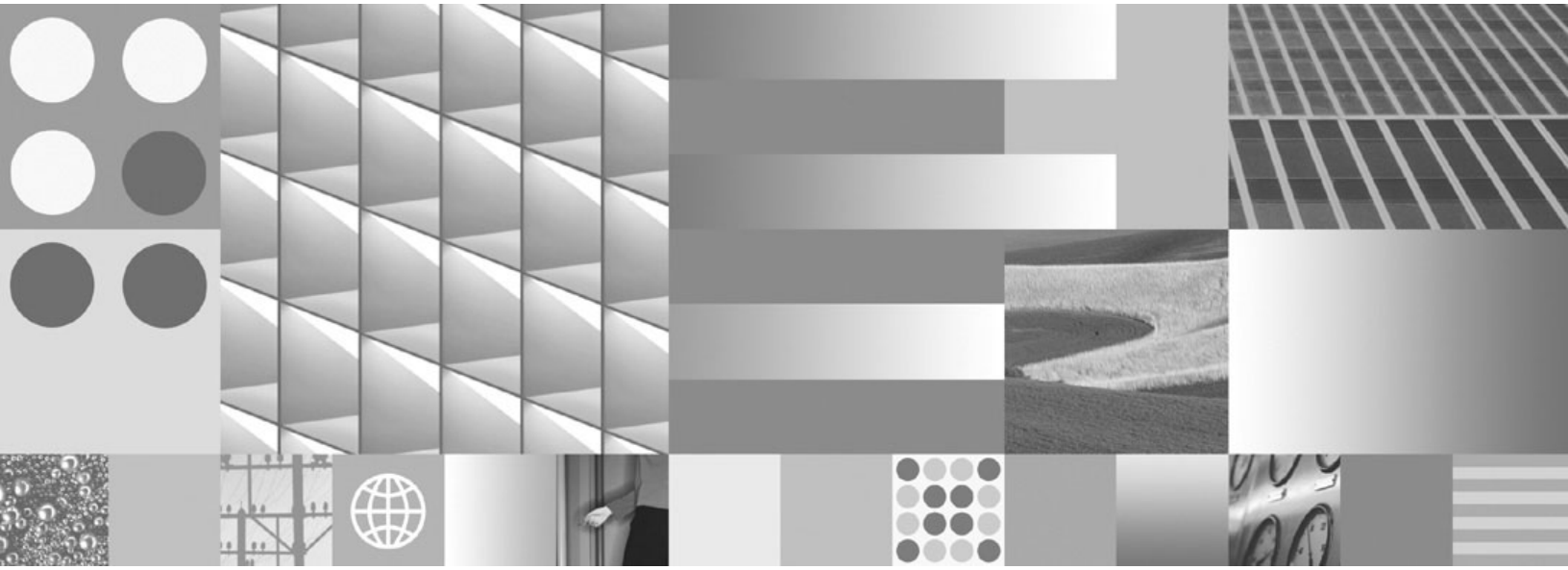


Řízení výkonu systému IBM Informix Dynamic Server

G229-1393-00

IBM Informix

Verze 11.1



Řízení výkonu systému IBM Informix Dynamic Server

G229-1393-00

Note:
Než použijete tyto informace a popisovaný produkt, přečtěte si informace v části “Poznámky” na stránce C-1.

Obsah

Úvod

Obsah úvodní kapitoly

Úvodní kapitola shrnuje obsah této příručky a popisuje konvence, které v ní jsou používány.

Obsah příručky

Příručka obsahuje informace o tom, jak nakonfigurovat a provozovat IBM Informix Dynamic Server a zvýšit celkový výkon systému a dotazů SQL. Obsahuje informace o metodách a problémech ladění výkonu, které se týkají denní administrace databázového serveru a provádění dotazů. Ladění a měření výkonu zahrnuje širokou oblast výzkumu a praxe a může zahrnovat i informace nad rámec této příručky.

Informace v této příručce pomáhají provádět následující úlohy:

- Monitorování systémových zdrojů, které zásadním způsobem ovlivňují výkon.
- Identifikace databázových procesů, které tyto zdroje nejvíce využívají.
- Identifikace a monitorování dotazů, které jsou kritické pro výkon.
- Použití obslužných programů databázového serveru (zejména programů **onperf**, ISA a **onstat**) pro monitorování a ladění výkonu.
- Odstranění kritických míst výkonu následujícími způsoby:
  - Rozložením zatížení systémových zdrojů
  - Nastavením konfiguračních parametrů nebo proměnných prostředí databázového serveru
  - Nastavením uspořádání dat
  - Přidělením zdrojů dotazům pro podporu rozhodování
  - Vytvořením indexů pro rychlejší vyhledávání dat

Příručka také obsahuje úplný popis obslužného programu **onperf**.

Témata, která tato příručka neobsahuje

Pokusy vyrovnat zatížení serveru často vedou ke zvýšení výkonu. Tato vylepšení někdy mohou být výrazná. V některých situacích ale vyrovnání zátěže nestačí. Následující situace mohou vyžadovat zásah, který překračuje rozsah této příručky:

- Aplikační programy, které je třeba upravit tak, aby lépe využívaly zdroje databázového serveru nebo operačního systému.
- Aplikace, které se vzájemně ovlivňují a snižují výkon.
- Hostitelský počítač, který může způsobovat konflikty.
- Hostitelský počítač, jehož kapacita neodpovídá jeho zatížení.
- Potíže s výkonem sítě, které ovlivňují klienta, server nebo další aplikace.

Tyto situace nelze vyřešit pomocí ladění databáze. Pokud je ale databázový server správně nakonfigurován, je tyto potíže snadnější identifikovat a vyřešit.

Důležité: Přestože při posuzování výkonu je třeba vzít v úvahu kromě vylepšení doby odezvy a efektivního využití systémových zdrojů také spolehlivost a dostupnost dat, věnuje se příručka pouze doby odezvy a využití systémových zdrojů. Další informace o zvýšení spolehlivosti databázového serveru a dostupnosti dat naleznete v částech o přepínání, zrcadlení a vysoké dostupnosti v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*. Další informace o zálohování a obnovování naleznete v příručce *IBM Informix Backup and Restore Guide*.

Druhy uživatelů

Tato příručka je určena pro následující uživatele:

- Administrátory databází.
- Administrátory databázových serverů.
- Programátory databázových aplikací.
- Techniky pro záležitosti výkonu.

Předpokladem pro práci s touto příručkou jsou následující znalosti:

- Práce s počítačem, operačním systémem a jeho obslužnými programy.
- Částečná znalost práce s relačními databázemi nebo obecná znalost problematiky databází.
- Částečná znalost programování.
- Částečná znalost administrace databázového serveru, operačního systému či sítě.

Nemáte-li dostatečné zkušenosti s relačními databázemi, jazykem SQL a používaným operačním systémem, naleznete další informace v příručce *Úvodní příručka serveru IBM Informix Dynamic Server* pro svůj databázový server.

Softwarové závislosti

Tato příručka předpokládá, že používáte IBM Informix Dynamic Server verze 11.1.

Předpoklady pro národní prostředí

Produkty IBM Informix podporují mnoho jazyků, národností a znakových sad. Veškeré informace o znakových sadách, třídění a reprezentaci číselných dat, měn, data a času jsou obsaženy v jediném prostředí nazvaném národní prostředí Global Language Support (GLS).

Tato příručka předpokládá použití národního prostředí U.S. 8859-1 English jako výchozího národního prostředí. Výchozí národní prostředí na platformách UNIX je **en_us.8859-1** (ISO 8859-1) nebo **en_us.1252** (Microsoft 1252) v systémech Windows. Toto národní prostředí podporuje formátovací konvence americké angličtiny pro datum, čas a měnu a také podporuje znakové sady ISO 8859-1 a Microsoft 1252, které obsahují znakovou sadu ASCII a mnoho dalších 8b znaků, např. é, è a ñ.

Použití znaků, které nejsou obsaženy ve výchozí znakové sadě, v datech či identifikátorech jazyka SQL nebo použití jiných pravidel pro třídění je možné až po nastavení příslušného národního prostředí.

Informace o nastavení jiného národního prostředí, příslušné syntaxi a dalších záležitostech týkajících se národního prostředí GLS naleznete v příručce *IBM Informix GLS User's Guide*.

Demonstrační databáze

Obslužný program DB–Access dodávaný s databázovým serverem IBM Informix obsahuje jednu nebo více demonstračních databází:

- Databáze **stores_demo** ilustruje použití relačního schématu obsahujícího údaje o fiktivním velkoobchodu se sportovním zbožím. Mnoho příkladů obsažených v příručkách serveru IBM Informix je založeno na databázi **stores_demo**.
- Databáze **superstores_demo** ilustruje použití objektového relačního schématu. Databáze **superstores_demo** obsahuje příklady rozšířených datových typů, dědičnosti typů a tabulek a uživatelských rutin.

Další informace o vytváření a naplňování demonstračních databází naleznete v příručce *IBM Informix DB–Access User's Guide*. Popis těchto databází a jejich obsahu naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Skripty, které lze použít k instalaci demonstračních databází, jsou uloženy v adresáři **$INFORMIXDIR/bin** na platformách UNIX nebo v adresáři **%INFORMIXDIR%\bin** v systémech Windows.

Nové vlastnosti serveru Dynamic Server verze 11.10

Verze 11.10 obsahuje nové vlastnosti, které zvyšují funkčnost databázového serveru a usnadňují jeho použití. Následující tabulka poskytuje informace o nových vlastnostech serveru IBM Informix Dynamic Server verze 11.10, které popisuje tato příručka. Úplný seznam nových funkcí této verze naleznete v příručce *IBM Informix Getting Started Guide*. Toto téma uvádí nové funkce, které se týkají této příručky.

Nové vlastnosti	Odkaz
Konfigurační parametr RTO_SERVER_RESTART umožňuje nastavit čas v sekundách, který bude mít server Dynamic Server k dispozici pro obnovu po potížích potom, co jej restartujete a uvedete do režimu online nebo klidového režimu.	“RTO_SERVER_RESTART” na stránce 5-38
Podpora omezeného blokování transakcí (označovaného jako neblokující kontrolní body). Toto umožňuje aplikacím zpracovávat transakce při zpracovávání kontrolního bodu. Tato vlastnost odstraňuje kontrolní body fuzzy. V předchozích verzích serveru Informix Dynamic Server bylo vyprazdňování LRU nastaveno velmi agresivně, aby byl zkrácen čas blokování transakcí kontrolního bodu. Nyní může být vyprazdňování LRU nastaveno méně agresivně.	“RTO_SERVER_RESTART” na stránce 5-38 a příručka *Příručka administrátora serveru IBM Informix Dynamic Server*
Schopnost používat jazyk SQL pro skládání odvozených tabulek do nadřazených dotazů.	“Odvozené tabulky složené do nadřazených dotazů” na stránce 10-16
Podpora vytváření vícenásobných spouštěčů BEFORE, FOR EACH ROW a AFTER pro jeden příkaz INSERT, UPDATE, DELETE nebo SELECT v tabulce nebo pohledu a podpora nového typu uživatelské rutiny nazvaného *spouštěcí rutina.*	“Provedení spouštěče” na stránce 10-32 další informace naleznete v příručce *IBM Informix Guide to SQL: Syntax*

Nové vlastnosti	Odkaz
Příkazy SQL, které explicitně nebo implicitně vytváří index B-stromu v nekrycím sloupci nebo sadě sloupců, automaticky vypočítají distribuci sloupce tabulky s hlavním klíčem.	“Aktualizace statistických údajů o všech použitých tabulkách” na stránce 9-21
Konfigurační parametr, který povoluje nebo zakazuje vložení statistických údajů dotazu do výstupu příkazu SET EXPLAIN.	“Část Query Statistics poskytuje informace pro ladění výkonu” na stránce 10-11
Nová volba LAST COMMITTED, která umožňuje příkazu SET ISOLATION COMMITTED READ snížit riziko konfliktů uzamykání při pokusu o čtení tabulky, a nový konfigurační parametr, který určuje, zda bude databázový server používat poslední potvrzenou verzi dat, dojde-li k uzamčení.	“Úroveň izolace potvrzené čtení” na stránce 8-6
Funkčnost pro vylepšení souběžnosti mezipamětí s vlastní pamětí a konfigurační parametr, který lze použít k určení informací o mezipaměti.	“Mezipaměti virtuálních procesorů CPU” na stránce 3-21
Zvýšený výkon předpřipravených souborů s přímým vstupem - výstupem	“Přímý vstup - výstup pro předpřipravené soubory pro bloky prostoru dbspace (pouze systém UNIX)” na stránce 5-3
Automatická rekompilace připravených příkazů	“Vynucení opětovné optimalizace, která má zabránit problému s indexem a předem vytvořeným příkazem, není-li povolen konfigurační parametr If AUTO_REPREPARE” na stránce 11-12

Konvence používané v dokumentaci

Tato část popisuje následující konvence použité v dokumentaci produktu IBM Informix Dynamic Server:

- Typografické konvence
- Konvence týkající se funkcí, produktů a platforem
- Diagramy syntaxe
- Konvence příkazového řádku
- Konvence kódu příkladů

Typografické konvence

Při zavádění nových termínů, znázorňování obsahu obrazovky, popisu syntaxe příkazů apod. používá tato příručka následující konvence.

Konvence	Význam
KLÍČOVÉ_SLOVO	Klíčová slova jazyků SQL, SPL a některých dalších programovacích jazyků se zobrazují velkými písmeny písma Serif.
kurzíva	V rámci textu jsou nové termíny znázorněny kurzívou. V příkladech syntaxe či kódu jsou hodnoty proměnných, které mají být zadány uživatelem, znázorněny kurzívou.
tučné písmo	Tučným písmem jsou znázorněny součásti programů (například třídy, události či tabulky), proměnné prostředí, názvy souborů, cest a prvky rozhraní (například ikony, položky v nabídce či tlačítka).
bezpatkové písmo	Bezpatkovým písmem je znázorněn text zobrazený daným produktem a text zadaný uživatelem.
KLÁVESA	Velkými písmeny písma sans serif jsou znázorněny klávesy, které by měly být stisknuty uživatelem.
>	Tento symbol označuje položku v nabídce. Například zápis “Zvolte **Nástroje> Možnosti**” znamená: Zvolte položku **Možnosti** v nabídce **Nástroje**.

Značky vlastností, produktů a platforem

Značky vlastností, produktů a platforem označují odstavce obsahující informace, které se vztahují pouze k danému objektu. Příklady těchto značek:

Dynamic Server

Označuje informace týkající se pouze serveru IBM Informix Dynamic Server.

Konec Dynamic Server

Jen pro Windows

Označuje informace týkající se pouze prostředí OS Windows.

Konec Jen pro Windows

Toto označení se může vztahovat k jednomu či více odstavcům v rámci jedné části. Pokud se k určitému produktu či platformě váže celá část, je to příslušným textem vyznačeno v jejím záhlaví. Například:

Řazení tabulky (Windows)

Konvence použité v ukázkovém kódu

V celé příručce se vyskytují příklady kódu SQL. Pokud není uvedeno jinak, není daný kód specifický pro žádný konkrétní aplikační vývojový nástroj prostředí IBM Informix.

Příkazy SQL nejsou oddělovány středníky pouze v příkladech. Setkáte se například s následujícím příkladem:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

Tento kód SQL je nutné pro každý produkt upravit podle příslušných pravidel syntaxe. Například při použití produktu DB–Access je nutné oddělit příkazy středníky. Při použití rozhraní SQL API je nutné před každý příkaz předřadit EXEC SQL a navíc příkaz zakončit středníkem (či jiným příslušným oddělovačem).

Tip: Tři tečky v kódu příkladu znamenají, že text příkladu není kompletní a při použití je nutné jej doplnit. Všechny části důležité pro objasnění daného tématu jsou však v příkladu obsaženy.

Podrobné pokyny pro používání příkazů SQL v jednotlivých aplikačních vývojových nástrojích nebo rozhraních SQL API naleznete v příručce pro příslušný produkt.

Další dokumentace

Produktovou dokumentaci si můžete prohlížet, prohledávat a tisknout z Informačního centra IBM Informix Dynamic Server na webových stránkách http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp.

Další dokumentaci k serveru IBM Informix Dynamic Server a s ním souvisejícíh produktů (včetně poznámek k verzi, počítači a dokumentaci naleznete v online knihovně produktu na webových stránkách http://www.ibm.com/software/data/informix/pubs/library/. Produktovou dokumentaci můžete rovněž najít na disku CD Quick Start, které je součástí dodávky produktu (produktovou dokumentaci možno i instalovat).

Kompatibilita s oborovými standardy

Organizace American National Standards Institute (ANSI) a International Organization of Standardization (ISO) společně ustanovily sadu oborových standardů jazyka SQL (Structured Query Language). Produkty IBM Informix založené na jazyku SQL jsou plně kompatibilní se standardem SQL-92 Entry Level (vydáno jako ANSI X3.135-1992). Tento standard je shodný se standardem ISO 9075:1992. Navíc je mnoho vlastností databázových serverů IBM Informix kompatibilních se standardy SQL-92 Intermediate Level a Full Level a se standardy X/Open SQL Common Applications Environment (CAE).

IBM ocení veškeré připomínky

Velice si vážíme vašich připomínek, každé opravy nebo objasnění, které považujete v našich příručkách za užitečné a které nám pomohou vylepšit budoucí verze. Tyto připomínky by měly obsahovat:

- Název a verzi příručky, kterou používáte.
- Číslo části a strany.
- Vaše návrhy týkající se obsahu příručky.

Své připomínky nám zašlete na následující e-mailovou adresu:

docinf@us.ibm.com

Tato e-mailová adresa je rezervována pro informování o chybách a opomenutích v dokumentaci. S technickými problémy se obraťte na oddělení technické podpory IBM. Pokyny naleznete na webu oddělení technické podpory produktuIBM Informix na adrese http://www.ibm.com/planetwide/.

Veškeré návrhy jsou vítány.

Kapitola 1. Základní fakta o výkonu

Obsah kapitoly

Tato příručka obsahuje informace týkající se problémů s laděním výkonu a metod administrace a provádění dotazů. Ladění a měření výkonu zahrnuje širokou oblast výzkumu a praxe a může zahrnovat i informace nad rámec této příručky. Podrobnosti o typech informací obsažených v této příručce a o tématech, které se nenacházejí v rámci této příručky, naleznete v části "Obsah příručky" na stránce xi.

Popis paralelního zpracování databázového serveru a aplikací, které tento databázový server používají, naleznete v příručce *Úvodní příručka serveru IBM Informix Dynamic Server.*

V této kapitole naleznete následující témata:

- Popis základního přístupu k měření a ladění výkonu
- Pokyny pro dosáhnutí přípustného počátečního výkonu v malé databázi
- Popis funkcí v rámci udržování dobrého výkonu
- Seznam témat, která nejsou obsažena v této příručce

Základní přístup k měření a ladění výkonu

Prvotní indikace problému s výkonem jsou většinou neurčitého charakteru - uživatelé mohou oznamovat, že systém reaguje pomalu. Uživatelé si také mohou stěžovat, že nemohou dokončit práci, že dokončení transakcí či zpracovávání dotazů trvá příliš dlouho, nebo že se aplikace během dne v určitých okamžicích zpomaluje. Chcete-li určit podstatu problému, je třeba změřit skutečné využití systémových zdrojů a vyhodnotit výsledky.

Uživatelé obvykle ohlašují problémy s výkonem v následujících situacích:

- Doby odezvy transakcí či dotazů jsou delší než obvykle.
- Prostupnost transakce nepostačuje k dokončení požadovaného pracovního zatížení.
- Dojde ke snížení propustnosti transakce.

Chcete-li zachovat optimální výkon databázových aplikací, vytvořte takový plán, pomocí kterého budete moci měřit systémový výkon, udržovat jej na odpovídající úrovni a v případě jeho poklesu provádět opravná měření. Specifická měření mohou obvykle pomoci předcházet problémům s výkonem a opravovat je. Včasným zjištěním problémů můžete předejít tomu, aby závažně postihly uživatele.

Výkon databázového serveru doporučujeme opakovaně optimalizovat. Jestliže pomocí následujících kroků nebude možné dosáhnout požadovaného zlepšení, je možné, že je problém způsoben nedostatečnými hardwarovými zdroji nebo neefektivním kódem v jedné nebo více klientských aplikací.

Postup optimalizace výkonu:

1. Určení cílů výkonu
2. Proveďte obvyklá měření využití zdrojů a databázové aktivity.
3. Identifikujte příznaky problémů s výkonem: nepřiměřené využití procesoru, paměti nebo disků.
4. Vylaďte konfiguraci operačního systému.
5. Vylaďte konfiguraci databázového serveru.
6. Optimalizujte konfiguraci bloků a prostoru dbspace (včetně umístění protokolů) a uspořádejte soubory, prostory a místa pro dočasné tabulky.
7. Optimalizujte umístění tabulek, velikost uspořádání oblastí a fragmentaci.
8. Vylepšete indexy.
9. Optimalizujte aktivity vstupu - výstupu na pozadí včetně protokolování, kontrolních bodů a čištění stránek.
10. Naplánujte zálohování a dávkování na hodiny mimo špičku.
11. Optimalizujte implementaci databázové aplikace.
12. Opakujte kroky 2 až 11.

Stručný úvod pro uživatele malé databáze

Tato část je určena pro uživatele, jejichž malá databáze se všemi tabulkami se nachází pouze na jednom disku a používají pouze jeden virtuální procesor.

Postup dosažení přípustného počátečního výkonu v malé databázi:

1. Vygenerováním statistických údajů tabulek a indexů poskytnete informace optimalizátoru dotazů, který tak bude moci zvolit plány dotazů s nejnižší odhadovanou nákladovostí.

   Tyto statistické údaje jsou minimem pro dosažení optimálního výkonu u jednotlivých dotazů. Pokyny naleznete v části “Aktualizace statických údajů, nejsou-li generovány automaticky” na stránce 13-7. Plán dotazů vybraný optimalizátorem pro jednotlivé dotazy naleznete v části “Zobrazení plánu dotazů” na stránce 13-3.
2. Chcete-li spustit dotaz v paralelním provedení, je nutné zapnout funkci Parallel Database Query (PDQ).

   Bez fragmentace tabulek mezi více disky nedojde paralelnímu prohledávání. V případě jediného virtuálního serveru nedojde k paralelním spojením ani k paralelnímu uspořádání. Priorita PDQ však může získat více paměti a uspořádání tak provést. Další informace popisuje Kapitola 12, “Paralelní databázový dotaz”, na stránce 12-1.
3. Chcete-li sloučit dotazové aplikace zpracovávání transakcí online (OLTP) a systém podpory rozhodování (DSS), můžete regulovat množství zdrojů, které může dlouhotrvající dotaz získat, aby tak nebyly ovlivněny transakce OLTP.

Více informací o ovládání zdrojů PDQ naleznete v části "Přidělování zdrojů paralelním databázovým dotazům" na stránce 12-7.

4. Sledováním relací a jejich sloučením do různých podrobností zvýšíte výkon jednotlivých dotazů.

   Další informace o jednotlivých nástrojích a podrobnostech o relaci, které mají být sledovány, naleznete v části "Monitorování využití paměti u jednotlivých relací" na stránce 13-30 a "Monitorování relací a jednotkových procesů" na stránce 13-34.

Cíle výkonu

Při určování cílů výkonu databázového serveru a jím podporovaných aplikací je třeba vzít v úvahu mnoho faktorů. Formulované cíle a priority výkonu aplikací by měly být srozumitelné a konzistentní, aby bylo možné jejich reálné a konzistentní dosažení. Při stanovování cílů výkonu je nutné vzít v úvahu následující otázky:

- Je nejvyšší prioritou maximalizace propustnosti transakcí, minimalizace doby odezvy konkrétních dotazů nebo dosažení nejlepšího možného sloučení těchto variant?
- S jakým druhem sloučení mezi jednoduchými transakcemi, rozšířenými dotazy pro podporu rozhodování a dalšími typy požadavků databázový server obvykle zachází?
- V jakém okamžiku jste ochotní vyměnit rychlost zpracovávání transakcí za dostupnost nebo riziko ztráty konkrétní transakce?
- Používá se tato instance databázového serveru v konfiguraci typu klient/server? Pokud ano, jaké jsou charakteristiky sítě, které ovlivňují jeho výkon?
- Jaké je očekávaný počet uživatelů?
- Je konfigurace omezena pamětí, prostorem na disku nebo zdroji procesoru?

Odpovědi na tyto otázky vám pomohou stanovit reálné cíle výkonu zdrojů a používaných aplikací.

Měření výkonu

Následující ukazatele popisují výkon systému zpracovávání transakcí:

- propustnost
- doba odezvy
- nákladovost na transakci
- využití zdrojů

Propustnost, doba odezvy a nákladovost na transakci jsou popsány v následujících částech. Využití zdrojů může mít v závislosti na kontextu jeden ze dvou významů. Tento pojem se může vztahovat k množství zdrojů, které konkrétní operace vyžaduje nebo používá, nebo se může vztahovat k aktuálnímu zatížení v komponentě systému. V prvně uvedeném významu se pojem používá ke srovnání přístupů dokončení dané úlohy. Pokud například daná operace uspořádání vyžaduje 10 megabajtů prostoru na disku, její využití zdrojů je vyšší než u jiné operace tohoto typu, která vyžaduje pouhých 5 megabajtů prostoru na disku. V případě druhého významu se pojem používá jako upozornění například na počet cyklů procesoru, které podléhají konkrétnímu dotazu během stanoveného časového intervalu.

Další informace o výkonnostních vlivech rozdílných úrovní zatížení systémových komponent naleznete v části "Využívání zdrojů a výkon" na stránce 1-7.

Propustnost

Propustnost měří celkový výkon systému. U systémů zpracovávání transakcí se propustnost měří obvykle v *počtu transakcí za sekundu* (TPS) nebo v *počtu transakcí za minutu* (TPM). Propustnost závisí na následujících faktorech:

- specifikace hostitelského počítače
- zahlcení zpracovávání v softwaru
- rozvržení dat v disku
- stupeň souběžnosti, který je podporován hardwarem i softwarem
- typy zpracovávaných transakcí

Měření propustnosti

Nejlepším způsobem, jak u aplikace změřit propustnost, je začlenit do ní kód, který protokoluje časové značky transakcí při jejich potvrzení. Pokud aplikace nepodporuje přímé měření propustnosti, můžete získat odhad zaznamenáním čísla příkazů COMMIT WORK, které databázový server protokoluje v daném časovém intervalu. Použitím obslužného programu **onlog** získáte seznam záznamů logického protokolu, které jsou zapsány do souborů protokolu. Pomocí informací z tohoto příkazu můžete zaznamenávat potvrzené transakce a operace spojené s vkládáním, odstraňováním a aktualizováním. Není však možné získat informace uložené ve vyrovnávací paměti logického protokolu, dokud nejsou zapsány do souboru protokolu.

Potřebujete-li rychlejší odezvu, odhad získáte použitím volby **onstat -p**. Pomocí příkazu SET LOG nastavíte protokolování u databází, které obsahují důležité tabulky, na režim bez vyrovnávací paměti. Úspěšné události příkazu COMMIT WORK nebo jiné události můžete do souboru protokolu zaznamenat pomocí důvěryhodného prověřovacího prostředku. Použití prověřovacího prostředku může zvýšit zahlcení spojené se zpracováváním prověřované události, což může snížit celkovou propustnost. Další informace o důvěryhodném prověřovacím prostředku naleznete v příručce *IBM Informix Security Guide*.

Standardní testy propustnosti

Dobře známé organizace jako např. Transaction Processing Performance Council (TPC) zajišťují standardní testy, které umožňují odpovídající porovnávání propustností mezi hardwarovými konfiguracemi a databázovými servery. IBM je aktivní členem TPC.

TPC zajišťuje následující standardizované testy pro měření propustnosti:

- TPC-A

  Tento test se používá pro porovnávání zpracovávání jednoduchých transakcí online (OLTP). Charakterizuje výkon systému zpracovávání jednoduchých transakcí a dává důraz na intenzivně aktualizované služby. Test TPC-A simuluje pracovní zatížení skládající se z několika uživatelských relací spojených sítí a vykazujících podstatnou aktivitu vstupu - výstupu disku.
- Test TPC-B

  Tento test se používá pro testování nejvyššího stupně zátěže propustnosti databáze. Používá stejné zatížení transakce jako test TPC-A, ale odstraňuje veškeré síťové a interaktivní operace, aby bylo měření propustnosti co nejkvalitnější.
- Test TPC-C

  Tento test se používá pro složité aplikace OLTP. Je odvozen z testu TPC-A a používá sadu aktualizací, transakcí určených jen pro čtení, dávkových operací, požadavků o odvolání transakce, soupeření zdrojů a dalších typů operací ve složité databázi, aby mohl zajistit lepší znázornění typických pracovních zatížení.
- Test TPC-D

Tento test měří výkonnost zpracovávání dotazů u skutečně velkých dotazů na základě časů dokončení. TPC-D je test podpory rozhodování vytvořený jako sada typických obchodních dotazů formulovaných jako dotazy jazyka SQL ve vztahu k velkým databázím (o velikosti v řádech gigabajtů či terabajtů).

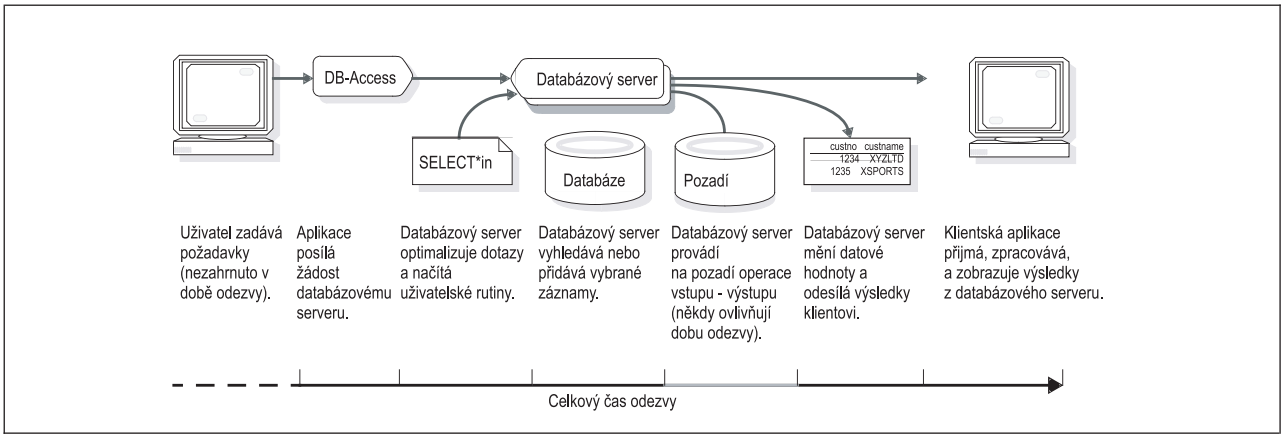
Protože má každá databázová aplikace své vlastní specifické pracovní zatížení, nelze pomocí testů TPC předem určit propustnost aplikace. Aktuálně dosažená propustnost závisí do značné míry na aplikaci.

Čas odezvy

Doba odezvy měří výkon jednotlivých transakcí nebo dotazů. Doba odezvy je obvykle doba, která uplyne od okamžiku, kdy uživatel zadá příkaz nebo aktivuje funkci, do okamžiku, kdy aplikace oznámí, že byly příkaz nebo funkce dokončeny. Doba odezvy typické Dynamic Server aplikace zahrnuje následující posloupnost akcí. Každá akce vyžaduje určité množství času. Doba odezvy nezahrnuje čas, který uživatel potřebuje k promyšlení a zadání dotazu nebo požadavku:

1. Aplikace předá dotaz databázovému serveru.
2. Databázový server provede optimalizaci dotazu a vyhledá uživatelem definované rutiny UDR). UDR zahrnuje rutiny SPL i externí rutiny. Další informace o rutinách UDR naleznete v části *IBM Informix User-Defined Routines and Data Types Developer's Guide*.
3. Databázový server vyhledává, přidává nebo aktualizuje příslušné dotazy a provádí operace disku s vstupem - výstupem, které s dotazem přímo souvisejí.
4. Databázový server provádí veškeré operace vstupu - výstupu běžící na pozadí (např. protokolování a čištění stránek), ke kterým dochází během doby, kdy dotaz nebo transakce čeká na vyřízení.
5. Databázový server vrátí výsledek aplikaci.
6. Aplikace zobrazí informace nebo vydá potvrzení, a poté uživateli vydá nový příkaz.

Část Obrázek 1-1 obsahuje diagram zobrazující ovlivnění celkové doby odezvy akcemi popsanými v krocích 1 až 6.



Obrázek 1-1. Komponenty doby odezvy u jediné transakce.

Doba odezvy a propustnost

doba odezvy a propustnost spolu souvisejí. Při zvyšování celkové propustnosti má doba odezvy průměrné transakce tendenci se snižovat. Přidělením neúměrného množství zdrojů dotazu však můžete snížit dobu odezvy u specifického dotazu na úkor celkové propustnosti. Celkovou propustnost můžete naopak zachovat omezením zdrojů, které databáze přidělí velkému dotazu.

Vyrovnání propustnosti a doby odezvy se projeví, až se pokusíte vyrovnat trvalou potřebu vyšší propustnosti transakce s okamžitou potřebou provést velký dotaz pro podporu rozhodování. Čím více zdrojů na dotaz použijete, tím méně bude možné zpracovávat transakce a tím větší může mít dotaz vliv na propustnost transakce. Naopak čím méně zdrojů na dotaz použijete, tím déle bude dotaz trvat.

Měření času odezvy

Pomocí jedné z následujících metod můžete změřit dobu odezvy dotazu nebo aplikace:

- příkazy časování operačního systému
- sledování výkonu operačního výkonu
- funkce časování v rámci aplikace

Příkazy časování operačního systému: Operační systém je obvykle vybaven obslužným programem, který můžete použít k načasování příkazu. Pomocí tohoto obslužného programu můžete změřit dobu odezvy příkazů jazyka SQL, které vydá příkazový soubor DB–Access.

Jen pro UNIX

Pokud disponujete příkazovým souborem, který provádí standardní sadu příkazů jazyka SQL, můžete pomocí příkazu **time** v mnoha systémech získat přesné časování těchto příkazů. Další informace o příkazových souborech naleznete v části *IBM Informix DB–Access User's Guide*. Následující příklad zobrazuje výstup příkazu **time** systému UNIX:

```
time commands.dba
...
4.3 real          1.5 user             1.3 sys
```

Výstup **čas** zobrazuje množství uplynulé doby (reálné), čas uživatelského procesoru a čas systémového procesoru. Pokud použijete prostředí C, první tři sloupce výstupu příkazu **time** prostředí C zobrazí položky v následujícím pořadí: uživatel, systém, uplynutá doba. Pokud doba systémového procesoru přesáhne 1/3 celkové uplynulé doby, aplikace provádí operace často nedokonale.

Příkaz **time** shromažďuje informace o aplikaci. Pomocí tohoto příkazu můžete vyvolat instanci aplikace, provést databázovou operaci a poté jeho ukončením získat údaje o času (viz následující příklad):

```
time sqlapp
   (enter SQL command through sqlapp, then exit)
10.1 real          6.4 user           3.7 sys
```

Můžete použít skript, pomocí kterého opakovaně spustíte stejný test, což vám umožní dosáhnout zhruba stejných výsledků za rozdílných podmínek. Vydělením uplynulé doby skriptu počtem databázových operací vykonávaných skriptem získáte odhad průměrné doby odezvy.

Konec Jen pro UNIX

Sledování výkonu operačního výkonu: Operační systémy jsou obvykle vybaveny aplikací pro sledování výkonu systému, pomocí které můžete změřit dobu odezvy dotazu nebo procesu.

Jen pro Windows

Pomocí aplikace Sledování systému můžete v systému Windows změřit následující doby:

- uživatelská doba
- doba procesoru

- uplynulá doba

Konec Jen pro Windows

Funkce časování v rámci aplikace: Většina programovacích jazyků je vybavena knihovní funkcí pro určitou část dne. Pokud máte přístup ke zdrojovému kódu, můžete k této funkci vložit dvojice volání a tak změřit uplynulou dobu mezi určitými akcemi.

Jazyk ESQL/C

Pokud je aplikace napsaná například v IBM Informix ESQL/C, můžete pomocí funkce **dtcurrent()** získat aktuální čas. Chcete-li změřit dobu odezvy, vyvoláním funkce **dtcurrent()** ohlásíte čas začátku transakce a poté čas jejího potvrzení.

Konec Jazyk ESQL/C

Uplynulá doba nemusí vždy v systému multiprogramování nebo v síťovém prostředí (kde jsou zdroje sdíleny mezi více procesy) odpovídat době provedení. Většina operačních systémů a knihoven C obsahuje funkce, které oznámí čas procesoru programu.

Nákladovost na transakci

Nákladovost na transakci je finanční ukazatel, který se obvykle používá při porovnávání celkových provozních nákladů v rámci aplikací, databázových serverů nebo hardwarových platforem.

Postup zjištění nákladovosti na jednu transakci:

1. Vypočítejte všechny náklady spojené s provozem aplikace.

   Tyto náklady mohou zahrnovat cenu za instalaci hardwaru a softwaru, provozní náklady včetně mezd a další výdaje.
2. Navrhněte celkový počet transakcí a dotazů pro životnost aplikace.
3. Vydělte celkové náklady celkovým množstvím transakcí.

Ačkoliv je toto měření užitečné při plánování a vyhodnocování, málokdy se týká denních problémů s dosahováním optimálního výkonu.

Využívání zdrojů a výkon

Běžná aplikace zpracovávající transakce podléhá během různých operačních cyklů rozdílným požadavkům. Nejvyšší zatížení během dne, týdne, měsíce a roku, stejně jako zátěž způsobená dotazy pro podporu rozhodování (DSS) nebo zálohovacími operacemi, mohou mít značný účinek na systém, který využívá téměř maximum své kapacity. Použitím přímých dat z historie, odvozených z konkrétního systému, můžete tento účinek stanovit.

Pomocí běžných měření pracovního zatížení a výkonu systému je možné předem určit nejvyšší zatížení a porovnat výsledky měření výkonu v různých okamžicích cyklu používání. Pravidelná měření vám pomohou pro aplikace databázového serveru vytvořit profil celkového výkonu. Tento profil je velmi důležitý při určování spolehlivého způsobu, jak zvýšit výkon.

Další informace o nástrojích měření, kterými je vybaven databázový server, naleznete v části “Nástroje databázového serveru” na stránce 2-3. Informace o nástrojích určených k měření účinků výkonu na systémové a hardwarové zdroje, kterými je vybaven váš operační systém, naleznete v části “Nástroje operačního systému” na stránce 2-2.

Využití je procento času, které aktuálně využívá určitá komponenta ve srovnání s celkovým časem, kdy je tato komponenta k dispozici. Pokud například procesor zpracovává transakce v jedné minutě po dobu 40 vteřin, jeho využití během tohoto intervalu je 67 procent.

Pravidelně zjišťujte a zaznamenávejte využití následujících systémových zdrojů:

- CPU
- Paměť
- Disk

Zdroj je pro výkon *kritický*, pokud je nadměrně používaný nebo pokud je jeho využití v nepoměru s využitím jiných komponent. Například disk můžete považovat za kritický nebo nadměrně používaný v případě, že je jeho využití 70 procent, zatímco využití ostatních disků v systému je 30 procent. Ačkoliv 70 procent neznamená, že je disk skutečně nadměrně používán, můžete výkon zvýšit přeskupením dat, čímž vyrovnáte žádosti vstupu - výstupu mezi celou sadou disků.

Měření využití zdrojů záleží na nástrojích operačního systému, pomocí kterých je možné oznamovat aktivitu systému a využití zdrojů. Jakmile identifikujete nadměrně používaný zdroj, můžete pomocí obslužných programů sledujících výkon shromáždit údaje a vyvodit závěry o databázových aktivitách, které mohou objasnit zátěž komponenty. Úpravou konfigurace databázového serveru nebo operačního systému můžete tyto databázové aktivity omezit nebo je rozložit mezi další komponenty. V některých případech bude možná nutné vyřešit problémy s výkonem zajištěním dalších hardwarových zdrojů.

Využití zdrojů

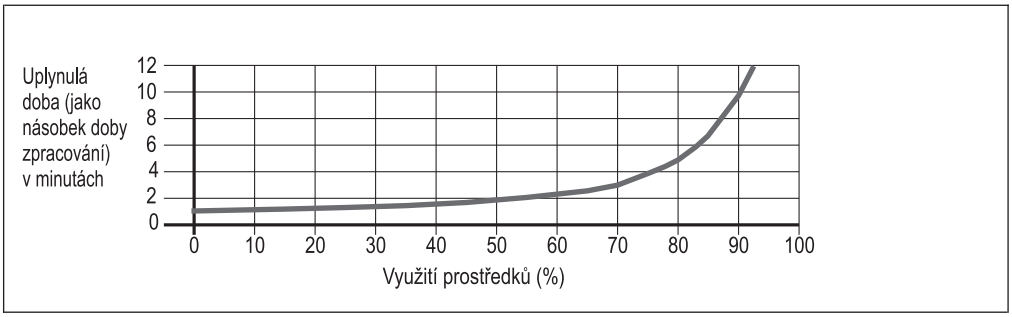
Vždy, když je systémový zdroj (např. procesor nebo konkrétní disk) zaneprázdněný transakcí nebo dotazem, nemůže zpracovávat jiné požadavky. Nevyřízené požadavky musejí před svým vyřízením vyčkat, dokud nebudou zdroje dostupné. Pokud je komponenta příliš zaneprázdněná, aby držela krok se všemi požadavky, nadměrně používaná komponenta začne vykazovat omezenou činnost. Čím vyšší je procento času, během kterého je zdroj zaneprázdněný, tím delší dobu musí operace čekat na své provedení.

Pomocí následujícího návodu můžete odhadnout dobu provozu požadavku, která je založená na celkovém využití komponenty obsluhující požadavek. Předpokládaná doba provozu zahrnuje dobu vynaloženou na čekání na zdroj a jeho následné použití v otázce. Považujte dobu provozu jako část doby odezvy, která je tvořená jedinou komponentou v rámci počítače (viz následující vzorec):

$S = P/(1-U)$

S	je předpokládaná doba provozu.
P	je doba zpracovávání, kterou operace vyžaduje, jakmile získá zdroj.
U	je využití zdroje (vyjádřeno v desetinných místech).

Jak je uvedeno v části Obrázek 1-2, doba provozu jediné komponenty se výrazně zvyšuje, dosáhne-li využití více než 70 procent. Například pokud transakce vyžaduje na zpracování danou komponentou 1 sekundu, můžete očekávat, že u komponenty s 50% využitím bude zpracovávání trvat 2 sekundy a u komponenty s 80% využitím 5 sekund. Dosáhne-li využití zdroje 90 procent, bude zpracovávání transakce trvat 10 sekund.



Obrázek 1-2. Doba provozu jediné komponenty jako funkce využití zdrojů.

Jestliže průměrná doba odezvy u typické transakce stoupne ze 2 nebo 3 sekund na 10 a více sekund, uživatelé tuto změnu zcela jistě zaznamenají a budou si stěžovat.

Důležité: Sledujte každý systémový zdroj, který zobrazuje využití více než 70 procent nebo každý zdroj, který projevuje příznaky nadměrného používání (viz následující části).

Berete-li v úvahu využití zdrojů, rozmyslete si také, zda je zvyšování velikosti stránky standardního nebo dočasného prostoru dbspace v daném prostředí užitečné. Pokud chcete, aby byla délka klíče větší, než je možné pro výchozí velikost stránky standardního nebo dočasného prostoru dbspace nastavit, můžete velikost této stránky zvýšit.

Využití procesoru

Použitím postupu z předchozí části můžete odhadnout dobu odezvy intenzivně zatíženého procesoru. Velké využití procesoru však nemusí vždy ukazovat na problém s výkonem. Procesor provádí veškeré výpočty potřebné ke zpracovávání transakcí. Čím více výpočtů týkajících se transakcí procesor v daném období provede, tím vyšší bude propustnost. Dokud je propustnost transakcí vysoká a vůči využití procesoru proporciální, velké využití procesoru naznačuje, že je počítač používán v plné míře.

Pokud je však na druhou stranu využití procesoru vysoké, ale propustnost transakcí nedrží stejné tempo, zpracovává procesor transakce neefektivně, nebo je zaneprázdněný aktivitou, která přímo nesouvisí s jejich zpracováváním. Cykly procesoru jsou přesměrovány na vnitřní řídící úlohy (např. správa paměti). Následující aktivity je možné snadno odstranit:

- Velké dotazy, které mohou být lépe naplánovány na dobu mimo špičku.
- Nesouvisející uživatelské rutiny, které lze lépe provádět na jiném počítači.

Pokud se doba odezvy zvýší do té míry, že budou prodlevy nepřijatelné, procesor může být zahlcený. Zátěž transakcí může být tak vysoká, že ji počítač nemusí zvládat spravovat. Dlouhá doba odezvy může také naznačovat, že procesor zpracovává transakce neefektivně nebo že došlo k přesměrování jeho cyklů.

Pokud je využití procesoru vysoké, může podrobná analýza databázového serveru odhalit všechny neefektivní zdroje, které jsou přítomny v důsledku nesprávné konfigurace. Další informace o analyzování aktivity databázového serveru naleznete v části “Nástroje databázového serveru” na stránce 2-3.

Využití paměti

Ačkoliv je pravidlo pro odhadování doby provozu paměti stejné, jako v části “Využívání zdrojů a výkon” na stránce 1-7, pro odhad účinku výkonu na využití paměti je nutné použít odlišný vzorec, než který používáte u jiných systémových komponent. Paměť není spravována jako jediná komponenta (jako např. procesor nebo disk), ale jako soubor malých komponent, nazývaných *stránky*. Velikost běžné stránky v paměti se může v závislosti na

operačním systému pohybovat v rozmezí od 1 do 8 kilobajtů. Počítač s 64 megabajty paměti a velikostí stránky o 2 kilobajtech obsahuje přibližně 32 000 stránek.

Jestliže operační systém potřebuje pro proces přidělit paměť, čistí všechny nepoužívané stránky v rámci paměti, které najde. Pokud volné stránky neexistují, systém správy paměti musí zvolit takové stránky, které jsou používané jinými procesy a u kterých je velice nepravděpodobné, že budou během krátkého období potřebné. Pro vybrání těchto stránek jsou vyžadovány cykly procesoru. Proces vyhledávání těchto stránek se nazývá *prohledávání stránek*. Je-li vyžadováno prohledávání stránek, zvýší se využití procesoru.

Systémy správy paměti obvykle pomocí algoritmu *naposledy použité* vybírají stránky, které lze zkopírovat na disk. Tyto stránky potom uvolňují, aby je mohly používat jiné procesy. Když procesor dokončí identifikaci stránek, které si může přivlastnit, *uvolní* obrazy původních stránek zkopírováním jejich původních dat na vyhrazený disk. Disk nebo oddíl disku, ve kterém jsou uložené obrazy stránek, se nazývá *odkládací disk, odkládací prostor,* nebo *odkládací oblast*. Tato stránkovací aktivita vyžaduje cykly procesoru i operace vstupu - výstupu.

Obrazy stránek, které byly zkopírovány na odkládací disk, je nutné případně znovu obnovit, aby je mohly používat procesy, které je vyžadují. Pokud je k dispozici stále velmi malé množství volných stránek, je nutné stránkováním vytvořit prostor. Když paměť nebude postačovat zvýšené poptávce a stránkovací aktivita se zvýší, může aktivita dosáhnout bodu, ve kterém již bude procesor téměř zcela zaneprázdněn. Systém, který se nachází v takovém stavu, je *zahlcený*. Je-li počítač zahlcen, dojde k pozastavení veškeré užitečné činnosti.

Překročí-li stránkovací aktivita určitou prahovou hodnotu, zabraňují některé operační systémy zahlcení pomocí zvláštního algoritmu správy paměti. Tento algoritmus se nazývá *odkládání*. Pokud se systém správy paměti uchýlí k odkládání, vyhradí si najednou všechny stránky tvořící celkový obraz procesu, ne pouze stránku.

Každou operací uvolňuje odkládání více paměti. V průběhu odkládání musí být každý odložený proces opět přečten, což výrazně zvyšuje diskový vstup - výstup odkládacího zařízení a čas požadovaný k přepínání mezi procesy. Výkon je potom omezen na rychlost, při které mohou být data přenášena z odkládacího disku zpět do paměti. Odkládání je příznakem kriticky přetíženého systému se zhoršenou propustností.

Mnoho systémů poskytuje informace o stránkovací aktivitě, mezi kterými je zahrnutý počet provedených prohledávání stránek, počet stránek odeslaných z paměti (*uvolněných*) a počet stránek přivedených do paměti (*zavedených)*:

- Stránkování je kritickým faktorem, protože operační systém stránkuje pouze v případě, kdy již nemůže najít volné stránky.
- Vysoká míra prohledávání stránek slouží jako předčasný ukazatel toho, že využití paměti začíná být nečinné.
- Stránky ukončených procesů jsou na místě uvolněny a znovu použity, takže stránkovací aktivita neposkytuje přesnou reflexi zátěže paměti. Vysoká míra stránkování může vyplývat z vysoké míry procesu fluktuace s nepodstatným účinkem na výkon.

Pomocí následujícího vzorce vypočítáte předpokládanou prodlevu stránkování u dané úrovně využití procesoru a míru stránkování:

```
PD= (C/(1-U)) * R * T
```

PD	je prodleva stránkování.
C	je doba provozu procesoru pro transakci.
U	je využití procesoru (vyjádřeno v desetinných místech).

R je míra stránkování.

T je doba provozu pro odkládací zařízení.

Při zvyšování stránkování dochází také ke zvyšování využití procesoru - tato navýšení jsou složená. Pokud míra stránkování 10 za sekundu odpovídá 5 procentům využití procesoru, zvýšení míry stránkování na 20 za sekundu může zvýšit využití procesoru o dalších 5 procent. Další zvyšování stránkování by vedlo k ještě prudšímu zvýšení využití procesoru, dokud by předpokládaná doba provozu pro požadavky procesoru nezačala nepřijatelná.

Využití disku

Protože se každý disk chová jako samostatný zdroj, můžete pomocí následujícího základního vzorce odhadnout dobu provozu, která je podrobně popsaná v části “Využití zdrojů” na stránce 1-8:

```
S= P/(1-U)
```

Protože se však přenosová rychlost mezi disky liší, většina operačních systémů neinformuje o využití disku přímo. Namísto toho udávají počet datových přenosů za sekundu (v jednotkách velikosti stránky paměti operačního systému). Chcete-li porovnat zátěž na discích se shodnou přístupovou dobou, jednoduše porovnejte průměrný počet přenosů za sekundu.

Pokud znáte přístupovou dobu k danému disku, můžete k výpočtu využití disku použít počet přenosů za sekundu. Chcete-li tak učinit, vynásobte průměrný počet přenosů za sekundu přístupovou dobou k disku (viz informace o disku poskytnuté výrobcem). V závislosti na uložení dat na disku se mohou přístupové doby od údajů výrobce lišit. Chcete-li tuto variabilitu započítat, přidejte k výrobcem uvedenému údaji o přístupové době 20 procent.

Následující příklad zobrazuje postup výpočtu využití disku s přístupovou dobou 30 milisekund a průměrným počtem 10 přenosových požadavků za sekundu:

```
U =     (A * 1.2) * X
  =     (.03 * 1.2) * 10
  =     .36
```

U je využití zdroje (v tomto případě disku).

A je výrobcem uvedená přístupová doba (v sekundách).

X je počet přenosů za sekundu udávaný operačním systémem.

Pomocí údaje o využití můžete odhadnout dobu zpracovávání transakce v disku, která vyžaduje dané množství diskových přenosů. Chcete-li vypočítat dobu zpracovávání transakce v disku, vynásobte počet diskových přenosů průměrnou přístupovou dobou. Chcete-li zahrnout také variabilitu přístupové doby, připočítejte navíc ještě 20 procent:

```
P = D (A * 1.2)
```

P je doba zpracovávání v disku.

D je počet diskových přenosů.

A je výrobcem uvedená přístupová doba (v sekundách).

Dobu zpracovávání transakce, která vyžaduje například 20 diskových přenosů z disku s přístupovou dobou 30 milisekund, můžete vypočítat následujícím způsobem:

```
P =    20 (.03 * 1.2)
  =    20 * .036
  =    .72
```

Pomocí vypočtených údajů o době zpracovávání a využití můžete u konkrétního disku odhadnout předpokládanou dobu provozu vstupu - výstupu (viz následující příklad):

```
S =    P/(1-U)
  =    .72 / (1 - .36)
  =    .72 / .64
  =    1.13
```

Faktory, které ovlivňují využití zdrojů

Výkon aplikace databázového serveru závisí na níže uvedených faktorech. Při určování problémů s výkonem nebo při upravování systému je třeba tyto faktory vzít v úvahu:

- Hardwarové zdroje

  Jak již bylo zmíněno výše v této kapitole, mezi hardwarové zdroje patří procesor, fyzická paměť a subsystémy diskového vstupu - výstupu.

- Konfigurace operačního systému

  Databázový server závisí na tom, zda operační systém umožňuje přístup k zařízením na nízké úrovni, plánování procesů, komunikaci mezi procesy a důležité služby.

  Konfigurace operačního systému má přímý vliv na výkon databázového serveru. Jádro operačního systému zabírá podstatné množství fyzické paměti, kterou tak databázový server ani jiné aplikace nemohou použít. Je však nutné vyhradit adekvátní zdroje jádra, které může databázový server používat.

- Nastavení sítě a přenosu

  Aplikace závislé na síti kvůli komunikaci s databázovým serverem a systémy závislé na replikaci dat kvůli zachování vysoké dostupnosti podléhají omezení výkonu této sítě. Datové přenosy v rámci sítě jsou obvykle pomalejší než datové přenosy z disku. Prodlevy sítě mohou mít podstatný vliv na výkon databázového serveru a dalších aplikačních programů, které jsou spuštěné na hostitelském počítači.

- Konfigurace databázového serveru

  Charakteristika instance databázového serveru (např. počet virtuálních procesorů, velikost rezidentní a virtuální části sdílené paměti a počet uživatelů) hraje při určování kapacity a výkonu aplikací důležitou roli.

- Konfigurace prostoru dbspace, blobspace a bloku

  Následující faktory mohou ovlivnit čas, který databázový server potřebuje k provedení transakcí diskového vstupu - výstupu a ke zpracování transakcí:

  – Umístění kořenového prostoru dbspace, fyzických protokolů a prostorů dbspace s dočasnými tabulkami

  – Přítomnost nebo nepřítomnost zrcadlení

  – Použití zařízení s vyrovnávací pamětí operačního systému nebo bez ní

- Umístění databází a tabulek

  Umístění tabulek a fragmentů v rámci prostorů dbspace, izolace velmi často používaných fragmentů v oddělených prostorech dbspace a rozložení fragmentů mezi více prostorů dbspace může ovlivnit rychlost, jakou může databázový server vyhledat datové stránky a přenést je do paměti.

- Uspořádání prostoru tblspace a stanovování velikosti oblastí

  Strategie fragmentace a velikost a umístění oblastí mohou mít vliv na schopnost databázového serveru rychle vyhledávat data v tabulce. Nepoužívejte prokládané oblasti a přidělujte takové oblasti, které se mohou přizpůsobit nárůstu velikosti, a tak předejít problémům s rychlostí.

- Účinnost dotazů

Správné provedení dotazu a použití kurzoru může snížit zátěž, kterou způsobuje některá z aplikací nebo některý uživatel. Upozorněte uživatele a vývojáře aplikací, že ostatní uživatelé vyžadují přístup k databázi a že každá aktivita ovlivňuje zdroje, které jsou k dispozici ostatním.

- Plánování aktivit vstupu - výstupu běžících na pozadí

  Protokolování, kontrolní body, čištění stránek a další operace (např. zálohování nebo spouštění velkých dotazů pro podporu rozhodování) mohou způsobovat neustálé zahlcování a velké zatížení systému. Zálohování a dávkové operace plánujte vždy na dobu mimo špičku.

- Vzdálené operace typu klient/server a distribuovaná spojení

  Tyto operace mají významný vliv na výkon, a to obzvláště hostitelského počítače, který koordinuje distribuovaná spojení.

- Účinnost kódu aplikace

  Aplikační programy zavádějí do operačního systému, do sítě a do databázového serveru vlastní zatížení. Pokud tyto programy málo využívají systémových zdrojů, generují přílišný síťový přenos nebo vytvářejí nepotřebné soupeření v databázovém serveru, mohou zahájit problémy s výkonem. Vývojáři aplikací musejí správně použít kurzory a blokovací úrovně, aby tak zachovali kvalitní výkon databázového serveru.

Udržování dobrého výkonu

Výkon je určitým způsobem ovlivňován všemi uživateli systému: administrátorem databázového serveru, administrátorem databáze, návrháři aplikací a uživateli klientské aplikace.

Administrátor databázového serveru obvykle koordinuje aktivity všech uživatelů, aby byl zajištěn odpovídající celkový výkon systému. Administrátor operačního systému může například potřebovat překonfigurovat operační systém tak, aby se zvýšilo množství sdílené paměti. Vypnutí operačního systému, které je nutné při instalaci operačního systému, vyžaduje také vypnutí databázového serveru. Administrátor databázového serveru musí tuto dobu nečinnosti naplánovat a oznámit všem zasaženým uživatelům, kdy systém nebude k dispozici.

Administrátor databázového serveru by měl řídit následujícími doporučeními:

- Uvědomit si všechny aktivity související s výkonem, ke kterým může dojít.
- Poučit uživatele o důležitosti výkonu, o působení aktivit s ním souvisejících a o možnostech dosažení a zachování optimálního výkonu.

Správce databázového serveru by měl věnovat pozornost následujícím faktorům:

- Jakým způsobem tabulky a dotazy ovlivňují celkový výkon databázového serveru.
- Umístění tabulek a fragmentů.
- Jakým způsobem ovlivňuje distribuce dat mezi disky výkon.

Vývojáři aplikací by se měli řídit následujícími doporučeními:

- Pečlivě navrhovat aplikace, aby bylo možné využívat souběžnosti a řadicích zařízení poskytovaných databázovým serverem, nikoliv se pokoušet implementovat podobná zařízení v aplikaci.
- Zachováním minimálního rozsahu a trvání zamknutí předcházet soupeření mezi databázovými zdroji.
- Začlenit rutiny v rámci aplikací, které při dočasném povolení při běhu programu umožňují administrátorovi databázového serveru sledovat dobu odezvy a propustnost transakcí.

Uživatelé databáze by se měli řídit následujícími doporučeními:

- Věnovat pozornost výkonu a případné problémy okamžitě oznamovat administrátorovi databázového serveru.
- Při plánování velkých dotazů pro podporu rozhodování se chovat ohleduplně a v rámci co nejrychlejšího dokončení tohoto úkolu vyžadovat co nejméně zdrojů.

Kapitola 2. Monitorování výkonu a použité nástroje

Obsah kapitoly

Tato kapitola popisuje nástroje pro monitorování výkonu a obsahuje vzájemně propojené sekce v podkapitolách o interpretaci výsledků monitorování výkonu. Popis nástrojů vám usnadní rozhodování při výběru nástroje pro vytvoření historie výkonu, monitorování databázových zdrojů v naplánovaných časech nebo monitorování výkonu probíhající transakce nebo dotazu.

Druh dat, která je třeba shromáždit, závisí na druhu aplikací, které jsou spuštěny v systému. Příčiny potíží s výkonem v systémech OLTP (Online Transaction Processing) jsou jiné než v systémech, které se primárně používají pro aplikace s dotazy DSS (Decision Support System). Systémy se smíšeným použitím jsou náročnější na vyladění výkonu a vyžadují propracovanou analýzu příčin potíží s výkonem.

Vyhodnocení aktuální konfigurace

Před začátkem konfigurace databázového serveru vyhodnoťte výkon aktuální konfigurace. Chcete-li změnit některé charakteristiky databázového serveru, je třeba server odstavit, což může ovlivnit váš provozní systém. Některá nastavení konfigurace mohou nechtěně snížit výkon nebo způsobit jiné negativní vedlejší účinky.

Pracují-li databázové aplikace dostatečně dobře na to, aby naplnily očekávání uživatelů, vyhněte se častému nastavování, i kdyby tato nastavení mohla teoreticky zvýšit výkon. Jsou-li uživatelé spokojeni, přistupujte ke změnám konfigurace databázového serveru opatrně. Je-li to možné, před provedením změn v provozním systému použijte testovací instanci databázového serveru k vyhodnocení změn konfigurace.

K vyhodnocení aktuální konfigurace databázového serveru lze použít nástroje popsané v této kapitole.

Týkají-li se potíže s výkonem zálohovacích operací, můžete také zkontrolovat počet a přenosové rychlosti páskových jednotek. Ke snížení dopadu zálohovacích operací může být nutné změnit rozvržení nebo fragmentaci tabulek. Další informace o rozvržení disku a fragmentaci tabulek naleznete v částech Kapitola 6, "Úvahy o výkonu tabulek", na stránce 6-1 a Kapitola 7, "Úvahy o výkonu indexů", na stránce 7-1.

V konfiguracích typu klient/server je třeba brát v úvahu také výkon a dostupnost sítě. Hodnocení výkonu sítě je mimo rámec této příručky. Další informace o monitorování síťových aktivit a vylepšení dostupnosti sítě získáte od administrátora sítě nebo v dokumentaci síťové sady.

Vytvoření historie výkonu

Nainstalujete-li databázový server a začnete na něm spouštět aplikace, měli byste také začít s plánovaným monitorování využití zdrojů. Shromáždit data pro analýzu výkonu lze pomocí obslužných programů příkazového řádku, které jsou popsány v částech "Nástroje databázového serveru" na stránce 2-3 a "Nástroje operačního systému" na stránce 2-2, ve skriptech nebo dávkových souborech.

Význam historie výkonu

Vytvoření historie výkonu a profilu systému vyžaduje pravidelné pořizování snímků s informacemi o využití zdrojů. Zaznamenáváte-li např. využití procesoru, rychlost stránkování a přenosovou rychlost vstupu - výstupu různých disků v systému, můžete zjišťovat úrovně a intervaly vrcholného využití a velmi vytížené zdroje. Sledujete-li využití fragmentů, můžete určit, zda je schéma fragmentace správně nakonfigurováno. Podle potřeby konfigurace databázového serveru a spuštěných aplikací můžete sledovat využití dalších zdrojů.

S těmito informacemi můžete začít hledat příčinu potíží, budou-li si uživatelé stěžovat na pomalou odezvu nebo nízkou propustnost. Není-li historie k dispozici, je třeba začít sledovat výkon po výskytu potíží, a tak nemusí být možné zjistit, kdy a jak potíže vznikly. Identifikace potíží až po jejich výskytu významně zpomaluje řešení potíží s výkonem.

Zvolte nástroje popsané v následujících částech a vytvořte úlohy, které budou zaznamenávat historii využití diskových, paměťových, vstupně-výstupních a dalších zdrojů databázového serveru. V této kapitole jsou stručně popsány výstupy jednotlivých nástrojů, abyste mohli vybrat nástroje pro vytvoření historie výkonu.

Nástroje pro vytvoření historie výkonu

Sledujete-li výkon databázového serveru, můžete použít nástroje hostitelského operačního systému a obslužné programy příkazového řádku, které můžete spouštět v pravidelných intervalech pomocí skriptů nebo dávkových souborů. K monitorování kritických momentů výkonu při zpracování dotazů a transakcí lze také použít nástroje pro monitorování výkonu s grafickým rozhraním.

Nástroje operačního systému

Databázový server využívá operační systém hostitelského počítače, který poskytuje přístup k systémovým zdrojům, jako je procesor, paměť a různá vstupně-výstupní rozhraní a soubory na discích bez vyrovnávací paměti. Každý operační systém má vlastní sadu obslužných programů, které informují o míře a způsobu využití systémových zdrojů. Různé implementace některých operačních systémů mohou mít monitorovací obslužné programy

téhož názvu, ale s odlišnými volbami a odlišným zobrazením příslušných údajů.

Jen pro UNIX

Budete moci používat některé z následujících nástrojů ke sledování zdrojů typických pro operační systém UNIX.

Nástroj pro systémUNIX	Popis
vmstat	Zobrazuje statistiku virtuální paměti.
iostat	Zobrazuje statistiku využití vstupu - výstupu.
sar	Zobrazuje řadu statistických údajů o zdrojích.
ps	Zobrazuje informace o aktivních procesech.

Podrobnosti o monitorování zdrojů operačního systému naleznete v referenční příručce nebo administrační příručce systému.

Chcete-li v pravidelných intervalech zachycovat stav systémových zdrojů, použijte nástroje pro plánování, které jsou k dispozici v hostitelském operačním systému (např. nástroj **cron**) jako součást systému pro monitorování výkonu.

Konec Jen pro UNIX

Jen pro Windows

Se systémem Windows se dodává nástroj Sledování výkonu (**perfmon.exe**), který umožňuje monitorování zdrojů, jako je procesor, paměť, vyrovnávací paměť, procesy a jednotkové procesy. Nástroj Sledování výkonu také poskytuje grafy, výstrahy a výkazy a umožňuje ukládat informace do souborů protokolů pro pozdější analýzu.

Další informace o použití nástroje Sledování výkonu naleznete v příručkách operačního systému.

Konec Jen pro Windows

Nástroje databázového serveru

Databázový server poskytuje nástroje pro pořizování snímků s informacemi o konfiguraci a výkonu. Také poskytuje rozhraní SMI (system-monitoring interface) pro monitorování výkonu v aplikaci.

Budete-li tyto nástroje pravidelně používat, můžete vytvořit historický profil aktivity databáze, který lze porovnat s aktuálními daty o využití zdrojů operačního systému. Tato porovnání pomáhají nalézt, které aktivity databázového serveru mají největší dopad na využití systémových zdrojů. Pomocí těchto informací lze identifikovat a spravovat aktivity s velkým dopadem nebo nastavit konfiguraci databázového serveru nebo operačního systému.

Mezi nástroje a obslužné programy databázového serveru, které lze použít k monitorování výkonu, patří:

- IBM Informix Server Administrator (ISA)
- **onstat**
- **onlog**
- **oncheck**

- DB–Access a rozhraní SMI (system-monitoring interface)

Jen pro UNIX

- ON–Monitor
- **onperf**

Konec Jen pro UNIX

Pomocí příkazů **onstat**, **onlog** a **oncheck** vyvolaných prostřednictvím plánovací služby **cron** můžete v pravidelných intervalech sbírat informace o výkonu a vytvořit z nich historický profil výkonu aplikace na databázovém serveru. V následujících částech jsou tyto obslužné programy popsány.

Pomocí příkazů SQL SELECT můžete spouštět dotazy na rozhraní SMI (system-monitoring interface) přímo ve vaší aplikaci.

Tabulky SMI je kolekce tabulek a pseudotabulek v databázi **sysmaster**, které obsahují dynamicky aktualizované informace o činnosti databázového serveru. Databázový server tyto tabulky vytváří v paměti, ale nezaznamenává je na disk. Volby nástroje **onstat** z těchto tabulek SMI získávají informace.

Pomocí služby **cron** a skriptů jazyka SQL v programu DB–Access nebo pomocí obslužného programu **onstat** můžete v pravidelných intervalech pokládat dotazy na tabulky SMI. Další informace o skriptech jazyka SQL naleznete v příručce *IBM Informix DB–Access User's Guide*. Další informace o tabulkách SMI naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Tip: Tabulky SMI se liší od tabulek systémového katalogu. Tabulky systémového katalogu obsahují trvale uložené a aktualizované informace o každé databázi a jejích tabulkách (někdy se označují také jako “metadata” nebo “datový slovník”). Další informace o tabulkách SMI naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*. Další informace o tabulkách systémového katalogu naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Pomocí programu ON–Monitor můžete zkontrolovat aktuální konfiguraci databázového serveru. Další informace o programu ON–Monitor naleznete v příručce *IBM Informix Administrator's Reference*.

Pomocí programu **onperf** lze zobrazit aktivitu databázového serveru prostřednictvím správce oken Motif . Další informace o programu **onperf** naleznete v části Kapitola 14, “Obslužný program onperf v systému UNIX”, na stránce 14-1.

Informix Server Administrator: Program IBM Informix Server Administrator (ISA) je nástroj založený na prohlížeči, který zajišťuje webovou systémovou administraci celé řady databázových serverů IBM Informix. Nástroj ISA je první z nové generace víceplatformových nástrojů administrace založených na prohlížeči. Poskytuje přístup ke všem funkcím příkazového řádku databázového serveru Informix a zobrazuje výstup ve snadno srozumitelném formátu.

Nástroj ISA je obsažen na disku CD-ROM databázového serveru dodaném s produktem. Další informace o instalaci nástroje ISA naleznete v následujícím souboru na disku CD-ROM.

Operační systém	**Soubor**
UNIX	**/SVR_ADM/README**

Windows **\SVR_ADM\readme.txt**

Pomocí nástroje ISA lze prostřednictvím prohlížeče provádět následující běžné úlohy administrace databázového serveru:

- Dočasně nebo trvale měnit parametry konfigurace.
- Měnit režim databázového serveru na režimy online a offline a přechodné stavy.
- Upravovat informace o konektivitě v souboru **sqlhosts**.
- Kontrolovat prostory dbspace a sbspace, protokoly a další objekty.
- Spravovat logické a fyzické protokoly.
- Vyhodnocovat využití paměti a přidávání a uvolňování paměťových segmentů.
- Číst protokol zpráv.
- Zálohovat a obnovovat prostory dbspace a sbspace.
- Spouštět různé příkazy **onstat** k monitorování výkonu.
- Zadávat jednoduché příkazy SQL a kontrolovat schémata databáze.
- Přidávat a odstraňovat bloky a prostory dbspace a sbspace.
- Kontrolovat a spravovat uživatelské relace.
- Kontrolovat a spravovat virtuální procesory (VP).
- Používat nástroje High-Performance Loader (zavaděč HPL), **dbimport** a **dbexport**.
- Spravovat replikace Enterprise Replication.
- Spravovat server IBM Informix MaxConnect.
- Používat následující obslužné programy: **dbaccess**, **dbschema**, **onbar**, **oncheck**, **ondblog**, **oninit**, **onlog**, **onmode**, **onparams**, **onspaces** a **onstat**.

Můžete také zadat libovolný obslužný program systému Informix, příkaz shellu systému UNIX nebo příkaz systému Windows (například **oncheck -cd**; **ls -l**).

Obslužný program onstat: Pomocí programu **onstat** lze kontrolovat aktuální stav databázového serveru a sledovat jeho aktivity. Tento program zobrazuje velké množství informací o výkonu a stavu serveru, které jsou obsaženy v tabulkách SMI. Úplný seznam všech voleb programu **onstat** zobrazíte spuštěním příkazu **onstat - -**. Úplný seznam všech informací, které program **onstat** shromažďuje zobrazíte spuštěním příkazu **onstat-a**.

Tip: Informace o profilu zobrazené programem **onstat**, např. příkazem **onstat -p** se shromažďují od data, kdy byl databázový server spuštěn. Chcete-li vymazat statistiku profilu výkonu a vytvořit nový profil, spusťte příkaz **onstat -z**. Používáte-li příkaz **onstat -z** k vynulování statistiky pro historii nebo vyhodnocení výkonu, ujistěte se, zda ostatní uživatelé nespouští tento příkaz také v jiných intervalech.

V následující tabulce jsou vypsány volby programu **onstat**, které zobrazují obecné informace o výkonu.

Volba	Popis
onstat -p	Zobrazuje profil výkonu, který obsahuje počet čtení a zápisů, počet požadavků na zdroj, který nebyl k dispozici a různé další informace.
onstat -b	Zobrazuje informace o aktuálně používaných vyrovnávacích pamětích.
onstat -l	Zobrazuje informace o fyzických a logických protokolech.
onstat -x	Zobrazuje informace o transakcích včetně identifikátoru jednotkového procesu uživatele, který transakci vlastní.
onstat -u	Zobrazuje profil aktivity uživatelů, který obsahuje informace

	o jednotkových procesech uživatelů, včetně ID relace vlastníka jednotkového procesu a přihlašovacího jména.
onstat -R	Zobrazuje informace o společné oblasti vyrovnávací paměti, včetně velikosti stránky společné oblasti vyrovnávací paměti.
onstat -F	Zobrazuje statistiku čištění stránek, která obsahuje počet zápisů všech typů, které vyprazdňují stránky, na disk.
onstat -g	Vyžaduje další argument, který určí, jaké informace budu zobrazeny. Např. příkaz **onstat -g mem** zobrazí statistiky paměti.

Další informace o volbách, které zobrazují informace týkající se výkonu, naleznete v části "Monitorování zdrojů databázového serveru" na stránce 2-6. Seznam a vysvětlení argumentů příkazu **onstat -g** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Monitorování zdrojů databázového serveru

Monitorováním specifických zdrojů databázového serveru lze identifikovat kritická místa výkonu a potenciální příčiny potíží a vylepšit využití zdrojů a dobu odezvy.

Jeden z nejužitečnějších příkazů pro monitorování systémových zdrojů je příkaz **onstat -g** a mnoho jeho voleb. Program ISA používá příkaz **onstat** k zobrazení informací. V částech "Monitorování využití fragmentace" na stránce 9-24 a "Monitorování a ladění mezipaměti příkazů SQL" na stránce 4-27 naleznete mnoho příkladů voleb příkazu **onstat -g**.

Monitorování zdrojů, které ovlivňují využití procesoru

Následující zdroje databázového serveru ovlivňují využití procesoru:

- Jednotkové procesy
- Komunikace po síti
- Virtuální procesory

Pomocí následujících argumentů příkazu **onstat -g** lze sledovat jednotkové procesy.

Argument	Popis
act	Zobrazuje aktivní jednotkové procesy.
ath	Zobrazuje všechny jednotkové procesy.
	Jednotkové procesy sqlexec představují části relací klientů, hodnota rstcb odpovídá poli user příkazu **onstat -u**.
rea	Zobrazuje připravené jednotkové procesy.
sle	Zobrazuje všechny spící jednotkové procesy.
sts	Zobrazuje maximální a aktuální využití zásobníku každého jednotkového procesu.
tpf *tid*	Zobrazuje profil jednotkového procesu pro zadaný identifikátor *tid*.
	Je-li identifikátor *tid* roven 0, zobrazí tento argument profily všech jednotkových procesů.
wai	Zobrazuje čekající jednotkové procesy, včetně všech jednotkových procesů, které čekají na objekt mutex nebo podmínku nebo které jsou spící.

Pomocí následujících argumentů příkazu **onstat -g** lze sledovat síť.

Argument	Popis

ntd	Zobrazuje síťové statistické údaje jednotlivých služeb.
ntt	Zobrazuje časy uživatelů sítě.
ntu	Zobrazuje statistické údaje uživatelů sítě.
qst	Zobrazuje statistické údaje fronty.

Pomocí následujících argumentů příkazu **onstat -g** můžete monitorovat virtuální procesory.

Argument	Popis
glo	Zobrazuje globální informace o jednotkových procesech včetně informací o využití virtuálních procesorů, celkovém počtu relací a dalších globálních ukazatelích.
sch	Zobrazuje počet operací, spinů a čekání na semafor pro každý virtuální procesor.
spi	Zobrazuje zámky longspin, což jsou zámky spin získané virtuálními procesory potom, co byly uzamčeny více než 10000krát. Chcete-li omezit počet zámků longspin, snižte počet virtuálních procesorů, snižte zatížení počítače nebo (na některých platformách) použijte parametr *no-age* nebo parametr *afinita procesoru*.
wst	Zobrazuje statistické údaje čekání.

Monitorování využití paměti

Pomocí následujících argumentů příkazu **onstat -g** můžete monitorovat využití paměti. Celkové informace o paměti získáte, pokud u příkazů, které povolují nepovinné parametry *název tabulky*, *název společné oblasti* nebo *ID_relace*, tyto parametry vynecháte.

Argument	Popis
ffr *název společné oblasti* | *ID_relace*	Zobrazuje volné fragmenty zadané společné oblasti sdílené paměti nebo zadané relace.
dic *název tabulky*	Zobrazuje jeden řádek informací o každé tabulce uložené ve slovníku sdílené paměti. Zadáte-li jako parametr název konkrétní tabulky, budou zobrazeny interní informace SQL o této tabulce.
dsc	Zobrazuje jeden řádek informací o každém sloupci statistických údajů distribuce uložených v mezipaměti distribuce dat.
mem *pool name* | *session id*	Zobrazuje statistické údaje paměti pro společné oblasti, které jsou spojeny s relací. Vynecháte-li *název společné oblasti* nebo *ID_relace*, budou zobrazeny informace o společné oblasti pro všechny relace.
mgm	Zobrazuje informace o zdrojích správce přidělujícího paměť včetně těchto: • Hodnoty konfiguračních PDQ • Informace o paměti a prohledávání • Informace o zatížení, například počet dotazů čekajících na paměť, počet dotazů čekajících na prohledávání, počet dotazů čekajících na běžící dotazy s nižší prioritou PDQ a počtem dotazů čekajících na slot dotazu • Aktivní dotazy a počet dotazů na každé bráně

- Statistika volných zdrojů
- Statistika dotazů
- Počet zabránění cyklu zamknutí zdroje, které zobrazuje, kolikrát systém okamžitě aktivoval dotaz, aby zabránil potenciálnímu zablokováni
-

nsc *ID klienta* Zobrazuje stav sdílené paměti pro zadané ID klienta.

Vynecháte-li *ID klienta*, budou zobrazeny stavové oblasti všech klientů.

nsd Zobrazuje data síťové sdílené paměti pro vyzvané jednotkové procesy.

nss *ID_relace* Zobrazuje stav síťové sdílené paměti pro zadané ID relace.

Vynecháte-li ID relace, budou zobrazeny stavové oblasti všech relací.

prc Zobrazuje jeden řádek informací ke každé uživatelské rutině (rutině SPL nebo externí rutině napsané v programovacím jazyce C nebo Java) uložené v mezipaměti uživatelských rutin

seg Zobrazuje statistické údaje segmentů sdílené paměti.

Je zobrazen počet a velikost všech připojených segmentů.

ses *ID_relace* Zobrazuje využití pamětí pro zadané ID relace.

Vynecháte-li ID relace, bude zobrazeno využití paměti pro všechny relace.

ssc Zobrazuje jeden řádek informací pro každý dotaz uložený v mezipaměti příkazů SQL.

stm *ID_relace* Zobrazuje využití paměti každého příkazu SQL pro zadané ID relace.

Vynecháte-li ID relace, bude zobrazeno využití paměti pro všechny relace.

ufr *název společné oblasti* | *ID relace*
Zobrazuje přidělené fragmenty společné oblasti zadaného uživatele nebo zadané relace.

Další informace o příkazech **onstat -g** a ukázku výstupu naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Monitorování využití vstupu - výstupu disku

Pomocí následujících obslužných programů lze určit, zda jsou vstupně-výstupní operace disku dostatečně výkonné pro vaše aplikace:

- argumenty příkazu **onstat -g**
- ISA
- obslužný program **oncheck**

Monitorování vstupu - výstupu pomocí příkazu onstat -g

Pomocí následujících argumentů příkazu **onstat -g** můžete monitorovat využití vstupu - výstupu disku.

Argument	Popis
iof	Zobrazuje statistické údaje asynchronního vstupu - výstupu po blocích nebo souborech. Tento argument je podobný příkazu **onstat -d**, který ale zobrazuje také informace o neblokových souborech. Tento argument zobrazuje informace o dočasných prostorech dbspace a souborech řazení.
iog	Zobrazuje globální informace asynchronního vstupu - výstupu.

ioq	Zobrazuje statistické údaje fronty asynchronního vstupu - výstupu.
iov	Zobrazuje statistické údaje asynchronního vstupu - výstupu podle virtuálních procesorů.

Podrobnou případovou studii, která používá různé výstupy obslužného programu **onstat** naleznete v části Dodatek A, “Případové studie a příklady”, na stránce A-1.

Monitorování vstupu - výstupu pomocí programu ISA

Pomocí programu ISA lze monitorovat využití vstupu - výstupu disku. Program ISA využívá k zobrazování informací o relacích informace, které generují následující volby příkazového řádku obslužného programu **onstat**, jak uvádí následující tabulka.

Monitorované informace	Výběr v programu ISA	Příslušný příkaz onstat
Statistické údaje asynchronního vstupu - výstupu po blocích nebo souborech	**Performance > AIO > Disk I/O by Queue**	**onstat -g iof**
Globální informace asynchronního vstupu - výstupu	**Performance > AIO > Global Information**	**onstat -g iog**
Statistické údaje asynchronního vstupu - výstupu podle virtuálních procesorů	**Performance > AIO > Disk I/O by VP**	**onstat -g iov**
Statistické údaje fronty asynchronního vstupu - výstupu	**Performance > AIO > Disk I/O by File**	**onstat -g ioq**

Monitorování vstupu - výstupu pomocí obslužného programu oncheck

Operace vstupu - výstupu disku tvoří obvykle nejdelší část doby odezvy dotazu. Souvisle přidělený prostor na disku zvyšuje výkon sekvenčních operací vstupu - výstupu disku, protože databázový server může číst větší bloky dat a pomocí funkce dopředného čtení snížit počet těchto operací.

Obslužný program **oncheck** zobrazuje informace o úložných strukturách na disku, jako jsou bloky, prostory dbspace a blobspace, oblasti, datové řady, tabulky systémového katalogu a další. Pomocí programu **oncheck** lze také určit počet oblastí v tabulce nebo zjistit, zda tabulka zabírá souvislé místo nebo ne.

Následující volby programu **oncheck** poskytují informace, které se vztahují k souvislému místu a oblastem. Další informace o použití dalších voleb programu **oncheck** naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Volba	Informace
-pB	Jednoduchý velký objekt blobspace (data typu TEXT nebo BYTE). Další informace o použití této volby k určení efektivity stránky blobpage naleznete v části“Určení zaplnění stránky blobpage pomocí příkazu oncheck -pB” na stránce 5-15.
-pe	Bloky a oblasti. Další informace o použití této volby k monitorování oblastí naleznete v částech “Kontrola prokládání oblastí” na stránce 6-25 a “Odstranění prokládaných oblastí” na stránce 6-26.

Volba	Informace
-pk	Hodnoty klíčů indexu. Další informace o zvýšení výkonu této volby naleznete v části "Zvýšení výkonu při kontrolách indexů" na stránce 7-14.
-pK	Klíče indexu a ID řádků. Další informace o zvýšení výkonu této volby naleznete v části "Zvýšení výkonu při kontrolách indexů" na stránce 7-14.
-pl	Hodnoty klíčů listů indexu. Další informace o zvýšení výkonu této volby naleznete v části "Zvýšení výkonu při kontrolách indexů" na stránce 7-14.
-pL	Hodnoty klíčů listů indexu a ID řádků. Další informace o zvýšení výkonu této volby naleznete v části "Zvýšení výkonu při kontrolách indexů" na stránce 7-14.
-pp	Stránky podle tabulek nebo fragmentů. Další informace o použití této volby k monitorování prostoru naleznete v části "Úvahy o horním limitu oblastí" na stránce 6-24.
-pP	Stránky podle bloků. Další informace o použití této volby k monitorování oblastí naleznete v části "Úvahy o horním limitu oblastí" na stránce 6-24.
-pr	Kořenové rezervované stránky. Další informace o použití této volby naleznete v části "Odhad tabulek s řádky fixní velikosti" na stránce 6-6.
-ps	Místo využité inteligentními velkými objekty a metadaty v prostoru sbspace.
-pS	Místo využité inteligentními velkými objekty a metadaty v prostoru sbspace a charakteristika úložiště. Další informace o použití této volby k monitorování prostoru naleznete v části "Monitorování prostorů sbspace" na stránce 6-14.
-pt	Prostor využitý tabulkou nebo fragmentem. Další informace o použití této volby k monitorování prostoru naleznete v části "Odhad velikosti tabulky" na stránce 6-6.
-pT	Prostor využitý tabulkou včetně indexů. Další informace o použití této volby k monitorování prostoru naleznete v části "Úvahy o výkonu příkazů DDL" na stránce 6-36.

Další informace o použití programu **oncheck** k monitorování prostoru naleznete v části "Odhad velikosti tabulky" na stránce 6-6. Další informace o souběžném spuštění programu **oncheck** naleznete v části "Zvýšení výkonu při kontrolách indexů" na stránce 7-14. Více informací o programu **oncheck** naleznete v příručce*IBM Informix Administrator's Reference*.

Monitorování transakcí

Pomocí následujících obslužných programů lze monitorovat transakce:

- obslužný program **onlog**
- obslužný program **onstat**
- ISA

Obslužný program onlog

Obslužný program **onlog** zobrazuje všechny nebo vybrané části logického protokolu. Vstupy tohoto příkazu mohou být vybrané soubory protokolu, celý logický protokol nebo záložní páska s předchozími soubory protokolu. Obslužný program **onlog** pomáhá identifikovat problematické transakce nebo změřit aktivitu transakcí odpovídající období velkého využití, které je zjištěno pomocí pravidelných snímků aktivity databáze a spotřeby systémových zdrojů.

Chcete-li pomocí programu **onlog** číst soubory logického protokolu, které jsou stále na disku, postupujte opatrně, protože čtení zablokovaných souborů protokolu zastaví ostatní aktivity databázové. Vyšší bezpečnosti dosáhnete, pokud nejprve zálohujete soubory logického protokolu a potom čtete obsah zálohovaných souborů. Správným postupem můžete pomocí příkazu **onlog -n** omezit program **onlog** pouze na uvolněné soubory logického protokolu. Stav souborů logického protokolu lze zkontrolovat pomocí příkazu **onstat -l**. Další informace o programu **onlog** naleznete v příručce *IBM Informix Administrator's Reference*.

Monitorování transakcí pomocí obslužného programu onstat

Je-li propustnost transakcí příliš nízká, pomocí následujících voleb příkazu**onstat** můžete identifikovat, která transakce výkon snižuje.

Volba	Popis
onstat -x	Zobrazuje informace o transakci, např. počet blokovaných zámků a úroveň izolace.
onstat -u	Zobrazuje informace o každém uživatelském jednotkovém procesu.
onstat -k	Zobrazuje zámky blokované každou relací.
onstat -g sql	Zobrazuje poslední příkaz SQL spuštěný během této relace.

Monitorování transakcí pomocí programu ISA

Monitorováním transakcí zaznamenáte otevřené transakce a zámky, které tyto transakce obsahují. Pomocí programu ISA lze monitorovat transakce a uživatelské relace. Program ISA zobrazuje informace o relaci, které jsou generovány prostřednictvím voleb příkazu **onstat**, jak je uvedeno v následující tabulce. Klepnutím na tlačítko **Refresh** opětovně spustíte příkaz obslužného programu **onstat** a zobrazíte aktualizované informace.

Monitorované informace	Výběr v programu ISA	Příslušný příkaz onstat	Další informace
Statistické údaje o transakci, např. počet blokovaných zámků a úroveň izolace	**Users > Transaction**	**onstat -x**	“Zobrazení transakcí pomocí příkazu onstat -x” na stránce 13-39
Statistické údaje o uživatelských relacích	**Users > Threads**	**onstat -u**	“Zobrazení uživatelských relací pomocí příkazu onstat -u” na stránce 13-41
Statistické údaje o zámcích	**Performance > Locks**	**onstat -k**	“Zobrazení zámků pomocí příkazu onstat -k” na stránce 13-40
Relace se spuštěnými příkazy SQL	**Users > Connections > session-id**	**onstat -g sql** *ID_relace*	“Zobrazení relací provádějících příkazy jazyka SQL” na stránce 13-42

Monitorování relací a dotazů

Chcete-li monitorovat aktivitu databázového serveru, můžete zobrazit počet aktivních relací a množství zdrojů, které používají. Monitorování relací a jednotkových procesů je důležité pro relace, které provádějí dotazy, stejně jako pro relace, které provádějí vkládání, aktualizace a odstraňování. Některé z informací, které lze monitorovat pro relace a jednotkové procesy, umožňují určit, zda aplikace nevyužívá příliš mnoho zdrojů.

Monitorování využití paměti u jednotlivých relací

Pomocí následujících argumentů příkazu **onstat -g** lze získat informace o paměti pro každou relaci.

Argument	Popis
ses	Zobrazuje jednořádkový souhrn všech aktivních relací.
ses *ID_relace*	Zobrazuje informace o relaci pro zadané *ID relace*.
sql *ID_relace*	Zobrazuje informace SQL zadané relace. Vynecháte-li parametr *ID_relace*, budou zobrazeny souhrny všech relací.
stm *ID_relace*	Zobrazuje množství paměti využité každým příkazem SQL spuštěným v relaci. Vynecháte-li parametr *ID_relace*, budou zobrazeny informace o všech spuštěných příkazech.

Příklady a diskuze o obslužných programech příkazového řádku pro monitorování relací naleznete v části “Monitorování využití paměti u jednotlivých relací” na stránce 13-30 a “Monitorování relací a jednotkových procesů” na stránce 13-34.

Použití příkazu SET EXPLAIN

Pomocí příkazu SET EXPLAIN nebo direktivy EXPLAIN lze zobrazit plán dotazů, které optimalizátor vytvoří pro jednotlivé dotazy. Další informace naleznete v části “Zobrazení plánu dotazů” na stránce 13-3.

Kapitola 3. Vliv konfigurace na využití CPU

Obsah kapitoly

Tato kapitola pojednává o tom, jak může kombinace konfiguračních parametrů operačního systému a databázového serveru ovlivnit využití CPU. Tato kapitola pojednává také o parametrech, které přímo ovlivňují využití CPU, a popisuje jejich nastavení. Je-li to možné, jsou v této kapitole popsány pokyny a doporučení nastavení parametrů, která se mohou týkat různých typů zatížení.

Je-li ve stejném hostitelském počítači spuštěno více instancí databázového serveru, je výsledný výkon výrazně horší v porovnání se situací, kdy jedna instance databázového serveru spravuje více databází. Více instancí databázového serveru nedokáže vyrovnat zátěž

tak účinně, jako to dokáže jediný databázový server. Zabraňte vícenásobnému uložení v provozním prostředí, kde je rozhodující výkon.

Konfigurační parametry systému UNIX, které mají vliv na využití procesoru

Instalační médium databázového serveru obsahuje soubor Poznámky k počítači, který obsahuje doporučené hodnoty konfiguračních parametrů pro systém UNIX. Porovnejte hodnoty v tomto souboru se současnou konfigurací operačního systému. Informace o umístění souboru Poznámky k počítači naleznete v úvodu této příručky.

Následující paramtery systémuUNIX mají vliv na využití procesoru:

- Parametry semaforu
- parametry nastavující maximální počet otevřených deskriptorů souboru
- parametry konfigurace paměti

Parametry semaforu systému UNIX

Semafory jsou zdroje jádra, které mají typickou velikost 1 bajt. Semafory pro databázový server jsou přiděleny navíc spolu s libovolnými dalšími semafory, které přidělíte pro ostatní soubory programů.

Každá instance databázového serveru vyžaduje následující sadu semaforů:

- Jedna sada pro každou skupinu nejvýše 100 virtuálních procesorů spuštěných spolu s databázovým serverem
- Jedna sada pro každý další virtuální procesor, který případně dynamicky přidáte při spuštěném databázovém serveru
- Jedna sada pro každou skupinu 100 nebo méně uživatelských relací připojených prostřednictvím komunikačního rozhraní sdílené paměti

Tip: Pro nejlepší výkon doporučujeme přidělit dostatek semaforů pro dvojnásobek očekávaného počtu připojení **ipcshm**. Dále doporučujeme pomocí konfiguračního parametru *NETTYPE* nakonfigurovat vyzvané jednotkové procesy databázového serveru pro tento dvojnásobný počet připojení. Popis vyzvaných jednotkových procesů viz *IBM Informix Administrator's Guide*. Informace o konfiguraci vyzvaných jednotkových procesů viz “Konfigurace vyzvaných jednotkových procesů pomocí konfiguračního parametru NETTYPE” na stránce 3-12.

Obslužné programy jako například **onmode** používají připojení prostřednictvím sdílené paměti, a proto je třeba nakonfigurovat minimálně dvě sady semaforů pro každou instanci databázového serveru: jednu pro počáteční sadu virtuálních procesorů a druhou pro připojení prostřednictvím sdílené paměti, která používají obslužné programy databázového serveru. Konfigurační parametr operačního systému SEMMNI typicky určuje počet sad semaforů, které se mají přidělit. Informace o nastavení parametrů týkajících se semaforů naleznete v instrukcích pro konfiguraci operačního systému.

Konfigurační parametr operačního systému SEMMSL typicky určuje maximální počet semaforů v jedné sadě. Nastavte tento parametr na hodnotu nejméně 100.

Některé operační systémy vyžadují konfiguraci maximálního celkového počtu semaforů ve všech sadách, což typicky určuje konfigurační parametr operačního systému SEMMNS . Pomocí následujícího vzorce vypočtete celkový počet semaforů, které vyžaduje každá instance databázového serveru:

```
SEMMNS = poč_virt_proc + přid_virt_proc + (2 * uživatelé_shmem) + soub_obsl_programy
```

poč_virt_proc počet virtuálních procesorů spuštěných s databázovým serverem. Zahrnuje virtuální procesory CPU, PIO, LIO, AIO, SHM, TLI, SOC a ADM. (Popis těchto virtuálních procesorů viz *IBM Informix Administrator's Guide*.) Minimální hodnota je 15.

přid_virt_proc počet virtuálních procesorů, které zamýšlíte dynamicky přidat.

uživatelé_shmem
počet připojení prostřednictvím sdílené paměti, které povolíte pro tuto instanci databázového serveru.

soub_obsl_programy
počet souběžných obslužných programů databázového serveru, které lze připojit k této instanci. Je doporučeno počítat minimálně s šesti připojeními obslužných programů: dvěma pro program ON–Archive nebo ON–Bar a čtyřmi pro jiné obslužné programy, například ON–Monitor (pouze systém UNIX), **onstat** a **oncheck**.

Používáte-li soubory programů, které vyžadují další semafory k těm, které potřebuje databázový server, je nutné, aby konfigurační parametr SEMMNI obsahoval celkový počet semaforů, které potřebuje databázový server a ostatní soubory programů. Konfigurační parametr SEMMSL je třeba nastavit na nejvyšší počet semaforů v sadě vyžadovaný libovolným souborem programů. U systémů vyžadujících konfigurační parametr SEMMNS získáte přijatelnou hodnotu vynásobením parametru SEMMNI hodnotou parametru SEMMSL.

Parametry popisovače souboru systému UNIX

Některé operační systémy vyžadují určení omezení počtu popisovačů souboru, které může mít proces současně otevřeny. Toto omezení můžete nastavit pomocí konfiguračního parametru operačního systému, obvykle parametrem NOFILE, NOFILES, NFILE nebo NFILES. Počet otevřených popisovačů souboru, které každá instance databázového serveru potřebuje, záleží na počtu bloků v databázi, počtu spuštěných virtuálních procesorů a na počtu síťových připojení, které musí instance databázového serveru podporovat.

Počet popisovačů souboru vyžadovaný pro instanci databázového serveru vypočítáte podle následujícího vzorce:

```
NFILES = (bloky * NUMAIOVPS) + NUMCPUVPS + síťová_připojení
```

bloky počet bloků, které mají být nakonfigurovány.

síťová_připojení počet síťových připojení, zadaný v některém z následujících umístění:

- Soubor **sqlhosts** nebo registr
- Konfigurační položky NETTYPE

Síťová připojení zahrnují všechna připojení kromě těch, které jsou určeny jako typ připojení **ipcshm**.

Každý otevřený popisovač souboru má přibližně stejnou délku jako celé číslo v jádru. Přidělení dalších popisovačů souboru je levným způsobem, jak umožnit růst počtu bloků nebo připojení v systému.

Konfigurační parametry paměti v systému UNIX

Konfigurace paměti v operačním systému může ovlivnit ostatní zdroje, včetně CPU a vstupu - výstupu. Nedostatek fyzické paměti pro celkovou zátěž systému může vést ke zhroucení, popis viz “Využití paměti” na stránce 1-9. Nedostatek paměti pro databázový server může mít za následek nadměrnou aktivitu při správě vyrovnávací paměti. Další informace o konfiguraci paměti viz “Konfigurace sdílené paměti UNIX” na stránce 4-5.

Konfigurační parametry systému Windows, které mají vliv na využití procesoru

Součástí distribuce Dynamic Server je soubor s poznámkami k verzi, který obsahuje doporučené hodnoty konfiguračních parametrů pro Dynamic Server v systému Windows. Porovnejte hodnoty v tomto souboru s aktuálním nastavením v souboru ONCONFIG. Cestu k souboru s poznámkami k verzi naleznete v úvodu této příručky.

Dynamic Server běží na pozadí. Chcete-li dosáhnout nejlepšího výkonu, dejte stejnou prioritu aplikacím v popředí i v pozadí.

Chcete-li v systému Windows změnit prioritu aplikací v popředí a pozadí, přejděte na položky **Start > Nastavení > Ovládací panely,** klepněte na ikonu **Systém** a klepněte na **kartu Upřesnit.** Klepněte na tlačítko **Možnosti výkonu**a vyberte klepněte na přepínač **Aplikace** nebo **Služby na pozadí**.

Konfigurace paměti v operačním systému může mít vliv na další prostředky, včetně CPU a vstupu - výstupu. Nedostatek fyzické paměti pro celkovou zátěž systému může vést ke zhroucení, popis viz “Využití paměti” na stránce 1-9. Nedostatek paměti pro Dynamic Server může mít za následek nadměrnou aktivitu při správě vyrovnávací paměti. Při nastavení hodnot **Virtuální paměť** na panelu **Systém** v okně **Ovládací panely** se ujistěte, že je k dispozici dostatek stránkovacího prostoru pro celkový objem fyzické paměti.

Konfigurační parametry a proměnné prostředí v souboru ONCONFIG, které mají vliv na využití CPU

Následující parametry v konfiguračním souboru databázového serveru mají značný vliv na využití CPU:

- DS_MAX_QUERIES
- DS_MAX_SCANS
- FASTPOLL
- MAX_PDQPRIORITY
- MULTIPROCESSOR
- NETTYPE
- OPTCOMPIND
- SINGLE_CPU_VP
- VPCLASS (NUMAIOVPS, NUMCPUVPS, NOAGE)
- VP_MEMORY_CACHE_KB

Následující části popisují vliv těchto konfiguračních parametrů na výkon. Další informace o konfiguračních parametrech databázového serveru naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Následující proměnné prostředí mají vliv na využití CPU:

- **OPTCOMPIND**
- **PDQPRIORITY**
- **PSORT_NPROCS**

Je-li proměnná prostředí **OPTCOMPIND** nastavena v prostředí klientské aplikace, ukazuje preferovaný způsob provádění spojovacích operací. Tato proměnná potlačuje hodnotu, která

je nastavena konfiguračním parametrem OPTCOMPIND. Podrobnosti o výběru preferované metody spojení viz “Optimalizace přístupových metod pomocí konifugračního parametru OPTCOMPIND” na stránce 3-9.

Je-li proměnná prostředí**PDQPRIORITY** nastavena v prostředí klientské aplikace, určuje limit využití virtuálního procesoru CPU (v procentech) limit využití sdílené paměti a dalších zdrojů, které lze přidělit dotazu spuštěnému klientem.

Klient může k nastavení hodnoty priority PDQ v SQL použít také příkaz SET PDQPRIORITY. Skutečná procentní část přidělená dotazu závisí na faktoru, který je nastaven konfiguračním parametrem MAX_PDQPRIORITY. Další informace o omezení zdrojů, které lze přidělit dotazu, viz “Omezení zdrojů PDQ v dotazech pomocí konfiguračního parametru MAX_PDQPRIORITY” na stránce 3-10.

Je-li proměnná **PSORT_NPROCS** nastavená v prostředí klientské aplikace, označuje počet paralelních jednotkových procesů řazení, které může aplikace využít. Databázový server zavádí u všech aplikací omezení na nejvýše 10 jednotkových procesů řazení na jeden dotaz. Další informace o paralelním řazení a proměnné **PSORT_NPROCS** viz “Konfigurace prostorů dbspace pro dočasné tabulky a soubory řazení” na stránce 5-7.

Další informace o proměnných prostředí, které mají vliv na databázové servery Informix, naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Určení třídy virtuálního procesoru pomocí konfiguračního parametru VPCLASS

Konfigurační parametr VPCLASS umožňuje určit třídu virtuálních procesorů, počet virtuálních procesorů, které má databázový server spustit pro určitou třídu a také maximální povolený počet.

Chcete-li spustit uživatelské rutiny (UDR), můžete definovat novou třídu virtuálních procesorů a tím izolovat provedení uživatelské rutiny od ostatních transakcí, které jsou prováděny na virtuálních procesorech CPU. Uživatelské rutiny obvykle píšete za účelem podpory uživatelských datových typů. Další informace o účelu uživatelských virtuálních procesorů naleznete v příručkách *IBM Informix Dynamic Server Administrator's Reference* a a *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Nechcete-li, aby uživatelská rutina měla vliv na normální zpracování uživatelských dotazů ve třídě CPU, můžete pomocí příkazu CREATE FUNCTION přiřadit rutinu k uživatelské třídě virtuálních procesorů. Název třídy určený parametrem VPCLASS musí odpovídat názvu, který je určený modifikátorem CLASS v příkazu CREATE FUNCTION. Další informace o příkazu CREATE FUNCTION naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Doporučujeme použít parametr VPCLASS jako alternativu k následujícím konfiguračním parametrům:

- AFF_SPROC
- AFF_NPROCS
- NOAGE
- NUMCPUVPS
- NUMAIOVPS

Při použití parametru VPCLASS je nutné explicitně odstranit tyto další parametry ze souboru ONCONFIG. Další informace o tom, které konfigurační parametry odstranit, naleznete v příručce *IBM Informix Administrator's Guide*.

Pokyny pro použití voleb **cpu** a **num** konfiguračního parametru VPCLASS naleznete v příručce "Nastavení počtu virtuálních procesorů CPU pomocí konfiguračního parametru VPCLASS".

Nastavení počtu virtuálních procesorů CPU pomocí konfiguračního parametru VPCLASS

Volby **cpu** a **num** konfiguračního parametru VPCLASS určují počet virtuálních procesorů CPU, které databázový server na počátku spustí. Nepřidělujte více virtuálních procesorů CPU, než je počet dostupných CPU pro jejich obsluhu.

Počet virtuálních procesorů CPU nastavíte podle následujících pokynů:

- U jednoprocesorových počítačů doporučujeme použít jeden virtuální procesor CPU.

  ```
  VPCLASS cpu,num=1
  ```

- U víceprocesorových systémů se čtyřmi a více CPU, které se primárně používají jako databázové servery, doporučujeme nastavit volbu **num** parametru VPCLASS na hodnotu o jednu menší, než je celkový počet procesorů. Pokud máte například čtyři procesory, použijte následující specifikaci:

  ```
  VPCLASS cpu,num=3
  ```

  Při použití tohoto nastavení je jeden procesor k dispozici pro spouštění obslužných programů databázového serveru nebo pro spouštění klientské aplikace.

- U víceprocesorových systémů, které primárně nepoužíváte na podporu databázových serverů, můžete začít s poněkud méně virtuálními procesory CPU, což umožní další aktivity systému, a v případě nutnosti můžete postupně přidat další.

U dvouprocesorových systémů je možné zlepšit výkon spuštěním dvou virtuálních procesorů CPU. Chcete-li otestovat, zda se výkon zlepší, nastavte v souboru ONCONFIG parametr NUMCPUVPS na hodnotu 1 a poté dynamicky přidejte virtuální procesor CPU za běhu příkazem **onmode -p**.

Zakázání stárnutí priority procesů virtuálních procesorů CPU pomocí konfiguračního parametru VPCLASS

Volba **noage** konfiguračního parametru VPCLASS umožňuje zakázat stárnutí priority procesů u virtuálních procesorů CPU databázového serveru s operačním systémem, který tuto funkci podporuje. Stárnutí priority znamená, že operační systém sníží prioritu dlouho spuštěných procesů, protože akumulují čas na zpracování. Je možné, že budete chtít zakázat stárnutí priority, protože může po určité době způsobit pokles výkonu procesů databázového serveru.

Součástí distribuce databázového serveru je soubor Poznámky k počítači, který obsahuje informace o tom, zda vaše verze databázového serveru tuto funkci podporuje. Informace o umístění souboru Poznámky k počítači naleznete v úvodu této příručky.

Nastavte volbu **noage** parametru VPCLASS v případě, že operační systém tuto funkci podporuje.

Určení afinity procesoru pomocí konfiguračního parametru VPCLASS

Volba **aff** u konfiguračního parametru VPCLASS určuje procesory, ke kterým se budou vázat virtuální procesory CPU nebo AIO. Když přiřadíte virtuální procesoru CPU konkrétnímu CPU, běží tento virtuální procesor pouze na daném CPU. Ostatní procesory mohou na tomto CPU také běžet.

Databázový server podporuje automatické vazby virtuálních procesorů CPU na procesory ve víceprocesorových hostitelských počítačích, které podporují afinitu procesoru. Součástí distribuce databázového serveru je soubor Poznámky k počítači, který obsahuje informace

o tom, zda vaše verze databázového serveru tuto funkci podporuje. Informace o umístění souboru Poznámky k počítači naleznete v úvodu této příručky.

Afinitu procesoru můžete využít k účelům, které jsou popsány v následujících částech.

Rozdělení vlivu výpočtu: Afinitu procesoru můžete využít k rozdělení vlivu výpočtu virtuálních procesorů CPU a dalších procesů. V počítačích vyhrazených pro databázový server dosáhnete přiřazením virtuálních procesorů na všechny CPU s výjimkou jediného maximálního využití CPU. V počítačích podporujících databázový server i klientské aplikace můžete navázat aplikace na určité CPU prostřednictvím operačního systému. Tímto postupem účinně vyhradíte zbývající CPU pro použití virtuálními procesory CPU databázového serveru, které navážete na zbývající CPU pomocí konfiguračního parametru VPCLASS. Nastavte volbu **aff** parametru VPCLASS na čísla CPU, na která chcete navázat virtuální procesory CPU. Následující nastavení parametru VPCLASS například přiřadí virtuální procesory CPU na procesory 4 a 7:

```
VPCLASS cpu,num=4,aff=4-7
```

Zadáte-li vyšší počet virtuálních procesorů CPU než je počet fyzických CPU, databázový server začne přiřazovat virtuální procesory CPU opět od počátečního CPU. Předpokládejme například následující nastavení parametru VPCLASS:

```
VPCLASS cpu,num=8,aff=4-7
```

Databázový server provede následující přiřazení:

- Virtuální procesor CPU číslo 0 k CPU 4
- Virtuální procesor CPU číslo 1 k CPU 5
- Virtuální procesor CPU číslo 2 k CPU 6
- Virtuální procesor CPU číslo 3 k CPU 7
- Virtuální procesor CPU číslo 4 k CPU 4
- Virtuální procesor CPU číslo 5 k CPU 5
- Virtuální procesor CPU číslo 6 k CPU 6
- Virtuální procesor CPU číslo 7 k CPU 7

Izolace virtuálních procesorů AIO od virtuálních procesorů CPU: V systému, kde je spuštěn databázový server a klientské (nebo jiné) aplikace můžete vázat virtuální procesory asynchronního vstupu - výstupu (AIO) na stejná CPU, na která vážete procesy ostatních aplikací prostřednictvím operačního systému. Tímto způsobem izolujete klientské aplikace a databázové operace vstupu - výstupu od virtuálních procesorů CPU. Tato izolace může být zvláště užitečná v situaci, kdy jsou klientské procesy používány pro zadávání dat nebo jiné operace, u kterých je nutné čekat na vstup od uživatele. Aktivita virtuálních procesorů AIO obvykle přichází v rychlých dávkách následovaných obdobími nečinnosti, takže můžete často promíchat klientské operace a operace vstupu - výstupu bez toho, aby se příliš ovlivňovaly.

Vázání virtuálního procesoru CPU na procesor nezabraňuje dalším procesům ve spouštění na daném procesoru. Procesy aplikací (nebo jiné procesy), které nejsou vázány na konkrétní CPU, se mohou volně spouštět na libovolném dostupném procesoru. V počítači vyhrazeném databázovému serveru můžete ponechat virtuální procesory AIO tak, aby běžely volně na libovolném procesoru, čímž snížíte prodlevy při databázových operacích, které čekají na vstup - výstup. Zvýšením priority virtuálních procesorů AIO můžete dále zvýšit výkon tím, že zajistíte rychlé zpracování dat okamžitě po tom, co přijdou z disku.

Vyhnutí se určitému CPU: Databázový server přiřazuje virtuální procesory CPU na CPU postupně a začíná s číslem CPU, které zadáte v tomto parametru. Budete se pravděpodobně chtít vyhnout přiřazení virtuálních procesorů CPU určitému CPU, které má specializovanou funkci týkající se hardwaru nebo operačního systému (například zpracování přerušení).

Chcete-li se vyhnout určitému CPU, nastavte volbu **aff** parametru VPCLASS na rozsah hodnot, který neobsahuje číslo daného CPU. Následující specifikace se například vyhne CPU číslo 3 a přiřadí virtuální procesory CPU na CPU číslo 3, 0 a 1:

```
VPCLASS cpu,num=3,aff=3-1
```

Nastavení počtu virtuálních procesorů AIO pomocí konfiguračního parametru VPCLASS

Volby **aio** a **num** parametru VPCLASS označují počáteční počet virtuálních procesorů AIO, které databázový server spustí. Pokud operační systém nepodporuje asynchronní vstup - výstup (KAIO), použije databázový server virtuální procesory AIO ke správě všech požadavků vstupu - výstupu databáze.

Doporučený počet virtuálních procesorů AIO závisí na tom, kolik disků podporuje konfigurace. Pokud na použité platformě *není* vstup - výstup KAIO implementován, měli byste přidělit jeden virtuální procesor AIO každému disku, který obsahuje databázové tabulky. Každému bloku, ke kterému databázový server často přistupuje, můžete přidat další virtuální procesor AIO.

Konfigurační parametr AUTO_AIOVPS umožňuje povolit databázovému serveru, aby zvětšil počet virtuálních procesorů AIO a jednotkových procesů čištění stránek, pokud server zjistí, že virtuální procesory nepostačují zátěži vstupu - výstupu.

Soubor Poznámky k počítači pro danou verzi databázového serveru ukazuje, zda operační systém podporuje procesy KAIO. Jsou-li procesy KAIO podporovány, v souboru Poznámky k počítači je popsáno, jak povolit KAIO v daném operačním systému. Informace o umístění souboru Poznámky k počítači naleznete v úvodu této příručky.

Pokud operační systém podporuje vstup - výstup KAIO, provádějí asynchronní požadavky vstupu - výstupu virtuální procesory procesoru místo virtuálních procesorů AIO. V tom případě nakonfigurujte pouze virtuální procesor AIO plus dva další virtuální procesory AIO pro každý souborový blok, který nepodporuje vstup - výstup KAIO.

Pokud používáte předpřipravené soubory a pokud povolíte přímý vstup - výstup pomocí konfiguračního parametru DIRECT_IO, můžete omezit počet procesorů. Pokud databázový server implementuje vstup - výstup KAIO a pokud je povolen přímý vstup - výstup, pokusí se server Dynamic Server používat vstup - výstup KAIO, takže patrně nebude třeba více než jeden virtuální procesor AIO. Dočasné prostory dbspace nepoužívají přímý vstup - výstup. Pokud máte dočasné prostory dbspace, potřebujete zřejmě více než jeden virtuální procesor AIO.

I když je povolen přímý vstup - výstup s konfiguračním parametrem DIRECT_IO, pokud souborový systém nepodporuje přímý vstup - výstup ani KAIO, stále potřebujete přidělit dva další virtuální procesory AIO pro každý aktivní blok prostoru dbspace, který vstup - výstup KAIO nepoužívá.

Cílem při přidělování virtuálních procesorů AIO je přidělit takové množství VP, aby byly fronty požadavků vstupu - výstupu krátké (tedy aby fronty obsahovaly co nejmenší počet požadavků vstupu - výstupu). Pokud zůstanou fronty požadavků na vstup - výstup stále krátké, jsou požadavky na vstup - výstup zpracovány stejným tempem, jakým přicházejí. Příkaz **onstat -g ioq** umožňuje monitorovat délku front vstupu - výstupu pro virtuální procesory AIO.

Přidělte dostatek virtuálních procesorů AIO tak, aby bylo možné zvládnout špičku v počtu požadavků na vstup - výstup. Obecně lze říci, že přidělení několika virtuálních procesorů AIO

navíc není na škodu. Chcete-li spustit další virtuální procesory AIO v okamžiku, kdy je databázový server v režimu online, použijte příkaz **onmode -p**. V režimu online nelze vypustit virtuální procesory AIO.

Nastavení konfiguračního parametru MULTIPROCESSOR při použití více virtuálních procesorů CPU

Provozujete-li více virtuálních procesorů CPU, nastavte parametr MULTIPROCESSOR na hodnotu 1. Nastavením parametru MULTIPROCESSOR na hodnotu 1 provede databázový server uzamknutí způsobem, který je vhodný pro víceprocesorový systém. Jinak nastavte tento parametr na hodnotu 0.

Počet virtuálních procesorů CPU je používán jako faktor pro určení počtu jednotkových procesů prohledávání pro dotaz. Dotazy jsou prováděny nejlépe v případě, že je počet jednotkových procesů násobkem počtu virtuálních procesorů CPU. Přidání nebo odstranění virtuálního procesoru CPU může zlepšit výkon u rozsáhlého dotazu, protože vytvoří rovnoměrné rozdělení jednotkových procesů prohledávání mezi virtuální procesory CPU. Pokud máte například 6 virtuálních procesorů CPU a prohledáváte 10 fragmentů tabulky, je možné, že se odezva zlepší po snížení počtu virtuálních procesorů CPU na 5, protože je lze rovnoměrně rozdělit mezi deset fragmentů. Můžete použít příkaz **onstat -g ath** pro monitorování počtu jednotkových procesů prohledávání na virtuální procesor CPU nebo se můžete pomocí příkazu **onstat -g ses** zaměřit na konkrétní relaci.

Nastavení konfiguračního parametru SINGLE_CPU_VP při použití jednoho virtuálního procesoru CPU

Je-li spuštěn pouze jeden virtuální procesor CPU, nastavte konfigurační parametr SINGLE_CPU_VP na hodnotu 1. Jinak jej nastavte na hodnotu 0.

Důležité: Nastavíte-li parametr SINGLE_CPU_VP na hodnotu *1*, musí být hodnota parametru NUMCPUVPS také *1*. Je-li druhý jmenovaný větší než *1*, databázový server se nespustí a zobrazí se následující chybová zpráva:

```
Nemůže být nenulová hodnota 'SINGLE_CPU_VP' a současně 'NUMCPUVPS' větší než 1
```

Poznámka: Databázový server zachází s třídami virtuálních procesorů definovanými uživatelem (tj. VP definované parametrem VPCLASS) jako kdyby šlo o virtuální procesory CPU. Z toho důvodu, nastavíte-li parametr SINGLE_CPU_VP na nenulovou hodnotu, nelze vytvářet žádné uživatelem definované třídy.

Nastavíte-li hodnotu parametru SINGLE_CPU_VP na 1, nelze přidávat virtuální procesory CPU ve chvíli, kdy je databázový server v režimu online.

Optimalizace přístupových metod pomocí konifugračního parametru OPTCOMPIND

Konfigurační parametr OPTCOMPIND pomáhá optimalizátoru dotazů zvolit vhodnou přístupovou metodu pro danou aplikaci. Při prověřování plánů spojení optimalizátorem ukazuje parametr OPTCOMPIND upřednostňovanou metodu provedení operací spojení dvojice seřazených tabulek.

Je-li parametr OPTCOMPIND nastaven na hodnotu 0, dává optimalizátor přednost existujícímu indexu (spojení vnořenou smyčkou) i v případě, že by prohledávání tabulek mohlo být rychlejší. Je-li parametr OPTCOMPIND nastaven na hodnotu 1 a úroveň izolace daného dotazu je nastavena na Repeatable Read, optimalizátor používá spojení vnořenou smyčkou.

Je-li parametr OPTCOMPIND nastaven na hodnotu 2 (výchozí), optimalizátor zvolí metodu čistě na základě nákladů, a to i přesto, že prohledávání tabulky mohou dočasně uzamknout celou tabulku. Další informace o parametru OPTCOMPIND a o různých metodách spojení viz "Vliv nastavení hodnoty OPTCOMPIND na plán dotazů" na stránce 10-21.

Chcete-li nastavit hodnotu proměnné OPTCOMPIND pro určité aplikace nebo uživatelské relace, nastavte proměnnou prostředí **OPTCOMPIND** pro tyto relace. Hodnoty této proměnné prostředí mají stejné rozsahy a sémantiku jako je tomu u hodnot konfiguračních parametrů.

Použití příkazu ENVIRONMENT OPTCOMPIND pro nastavení hodnoty proměnné OPTCOMPIND v rámci relace

Chcete-li nastavit nebo změnit hodnotu proměnné OPTCOMPIND v rámci relace, použijte příkaz SET ENVIRONMENT OPTCOMPIND.

Například můžete chtít použít různé hodnoty této proměnné pro různé druhy dotazů.

V případě dotazu DSS byste měli nastavit proměnnou OPTCOMPIND na hodnotu 2 nebo 1 a přesvědčit se, že úroveň izolace není nastavena na úroveň Repeatable Read. V případě dotazu OLTP můžete nastavit hodnotu na 0 nebo 1 v případě, že úroveň izolace není nastavena na Repeatable Read.

Hodnota zadaná prostřednictvím příkazu SET ENVIRONMENT OPTCOMPIND má přednost před výchozím nastavením určeným v souboru **ONCONFIG**. Výchozí nastavení OPTCOMPIND je obnoveno po ukončení aktuální relace. Provedenými příkazy SET ENVIRONMENT OPTCOMPIND nejsou ovlivněny žádné jiné uživatelské relace.

Další informace o příkazu SQL SET ENVIRONMENT a syntaxi příkazu SET ENVIRONMENT OPTCOMPIND naleznete v příručce *IBM Informix Guide to SQL: Syntax*

Omezení zdrojů PDQ v dotazech pomocí konfiguračního parametru MAX_PDQPRIORITY

Konfigurační parametr MAX_PDQPRIORITY omezuje procentní část zdrojů PDQ (parallel database query), které může dotaz využít. Pomocí tohoto parametru omezíte dopad rozsáhlých dotazů s intenzivním využitím CPU na propustnost transakcí.

Zadejte tento parametr jako celé číslo, které představuje procentní část následujících zdrojů PDQ, o které může dotaz požádat:

- Paměť
- virtuální procesory CPU
- Diskový vstup - výstup
- Jednotkové procesy prohledávání

Požádá-li dotaz o určitou procentní část zdrojů PDQ, databázový server přidělí z požadovaného objemu procentní část danou hodnotou MAX_PDQPRIORITY podle následujícího vzorce:

```
Přidělené zdroje = PDQPRIORITY/100 * MAX_PDQPRIORITY/100
```

Pokud například klient používá příkaz SET PDQPRIORITY 80 jako požadavek 80 procent zdrojů PDQ, ale parametr MAX_PDQPRIORITY je nastaven na hodnotu 50, databázový server přidělí klientovi pouze 40 procent zdrojů (50 procent požadavku).

V případě podpory rozhodování a online zpracování transakcí (OLTP) umožňuje nastavení parametru MAX_PDQPRIORITY administrátorovi databázového serveru řídit dopad

jednotlivých dotazů pro podporu rozhodování na výkon souběžného OLTP. Snižte hodnotu parametru MAX_PDQPRIORITY v případě, že chcete zpracování OLTP přidělit více zdrojů. Zvyšte hodnotu parametru MAX_PDQPRIORITY v případě, že chcete přidělit více zdrojů zpracování podpory rozhodování.

Další informace o řízení využití zdrojů PDQ viz "Přidělování zdrojů paralelním databázovým dotazům" na stránce 12-7.

Omezení dopadu dotazů s intenzivním využitím CPU na výkon pomocí konfiguračního parametru DS_MAX_QUERIES

Konfigurační parametr DS_MAX_QUERIES určuje maximální počet současně spuštěných dotazů pro podporu rozhodování. Jinak řečeno, parametr DS_MAX_QUERIES řídí pouze dotazy, jejichž priorita PDQ je nenulová. Dotazy s nízkou prioritou PDQ mají úměrně menší spotřebu zdrojů a proto může být současně spuštěn větší počet těchto dotazů. Pomocí parametru DS_MAX_QUERIES můžete omezit dopad dotazů s intenzivním využitím CPU na výkon.

Databázový server používá hodnoty DS_MAX_QUERIES a DS_TOTAL_MEMORY pro výpočet počtu kvant paměti, které přidělí dotazu. Další informace o způsobu přidělování paměti dotazům databázovým serverem naleznete v části "DS_TOTAL_MEMORY" na stránce 4-12.

Omezení počtu jednotkových procesů prohledávání PDQ, které mohou být spuštěny souběžně, pomocí konfiguračního parametru DS_MAX_SCANS

Konfigurační parametr DS_MAX_SCANS omezuje počet jednotkových procesů prohledávání PDQ, které mohou být spuštěny souběžně. Tento parametr zabraňuje tomu, aby byl databázový server zaplaven jednotkovými procesy prohledávání od více dotazů pro podporu rozhodování.

Pro výpočet počtu jednotkových procesů prohledávání přidělených dotazu použijte následující vzorec:

```
scan_threads  =                min (nfrags, (DS_MAX_SCANS * pdqpriority / 100
   * MAX_PDQPRIORITY / 100) )
```

nfrags počet fragmentů v tabulce s největším počtem fragmentů.

pdqpriority hodnota priority PDQ nastavená proměnnou prostředí **PDQPRIORITY** nebo příkazem SQL SET PDQPRIORITY.

Snížení počtu jednotkových procesů prohledávání může snížit dobu, po kterou rozsáhlý dotaz čeká ve frontě, zvláště pak v případě, že je více rozsáhlých dotazů odesíláno souběžně. Pokud je nicméně počet jednotkových procesů prohledávání nižší než *nfrags*, vlastní zpracování dotazu trvá déle.

Pokud je například u dotazu třeba prohledat 20 fragmentů tabulky, ale vzorcem *scan_threads* je umožněn začátek dotazu v okamžiku, kdy je k dispozici 10 jednotkových procesů prohledávání, každý jednotkový proces postupně prohledává dva fragmenty. Provedení dotazu trvá přibližně dvojnásobnou dobu oproti situaci, kdy by bylo použito 20 jednotkových procesů.

Konfigurace vyzvaných jednotkových procesů pomocí konfiguračního parametru NETTYPE

Konfigurační parametr NETTYPE konfiguruje vyzvané jednotkové procesy pro každý typ připojení, který instance databázového serveru podporuje. Pokud instance databázového serveru podporuje více než jedno rozhraní nebo protokol, je třeba zadat samostatný konfigurační parametr NETTYPE pro každý typ připojení.

Obvykle přidáváte samostatný NETTYPE parametr NETTYPE pro každý typ připojení, který je přidružený databázovému serveru s názvem dbservername. Seznam názvů dbservername získáte pomocí konfiguračních parametrů DBSERVERNAME a DBSERVERALIASES. Typy připojení přidružujete serverům s názvy dbservername v souboru **sqlhosts** nebo v registru. Informace o typech připojení a souboru **sqlhosts** naleznete v příručce *IBM Informix Administrator's Guide*.

Určení protokolu připojení

První položky NETTYPE u daného typu připojení se týká všech serverů, ke kterým je tento typ přidružen. Následující položky NETTYPE pro daný typ připojení jsou ignorovány. Položky NETTYPE jsou vyžadovány pro typy připojení používané pouze pro odchozí komunikaci a to i v případě, že tyto typy připojení nejsou uvedeny v souboru **sqlhosts** nebo v registru.

Jen pro UNIX

Následující protokoly se vztahují pouze k platformám UNIX:

- IPCSHM
- TLITCP
- IPCSTR
- SOCTCP

Konec Jen pro UNIX

Jen pro Windows

Následující protokoly se týkají platforem se systémem Windows:

- IPCNMP
- SOCTCP

Konec Jen pro Windows

Určení tříd virtuálních procesorů pro vyzvané jednotkové procesy

Každý vyzvaný jednotkový proces, který je nakonfigurovaný nebo přidaný dynamicky položkou NETTYPE je spuštěn v odděleném virtuálním procesoru. Vyzvaný jednotkový proces může být spuštěn jedné ze dvou tříd virtuálních procesorů: NET a CPU. Pro zajištění nejlepšího výkonu doporučujeme použít položku NETTYPE pro přiřazení pouze jednoho jednotkového procesu k třídě virtuálních procesoru CPU a další jednotkové procesy přiřadit virtuálním procesorům NET. Maximální počet vyzvaných jednotkových procesů přiřazených jednomu typu připojení nesmí překročit hodnotu NUMCPUVPS.

Určení počtu připojení a vyzvaných jednotkových procesů

Optimální počet připojení na vyzvaný jednotkový proces je přibližně 300 na jednoprocesorových počítačích a až 350 na víceprocesorových počítačích, i když se tato hodnota může měnit v závislosti na platformě a zatížení databázového serveru. Vyzvaný jednotkový proces může podporovat 1024 nebo i více připojení. Pokud je povolen

konfigurační parametr FASTPOLL, budete moci nakonfigurovat méně vyzvaných jednotkových procesů, ale bude třeba testovat výkon, aby byla zajištěna optimální konfigurace prostředí.

Každá položka NETTYPE konfiguruje počet vyzvaných jednotkových procesů pro určitý typ připojení, počet připojení na jeden jednotkový proces a třídu virtuálního procesoru, ve kterém jsou tyto jednotkové procesy spuštěny, pomocí polí oddělených čárkami. Uvnitř těchto polí a mezi nimi nejsou přípustné mezery.

```
NETTYPE connection_type,poll_threads,c_per_t,vp_class
```

connection_type
identifikuje kombinaci protokol-rozhraní, ke které jsou dané vyzvané jednotkové procesy přiřazeny. Toto pole obvykle nastavujete tak, aby odpovídalo poli *connection_type* u položky dbservername v souboru **sqlhosts** nebo v registru.

poll_threads
počet vyzvaných jednotkových procesů přiřazených k danému typu připojení. Nenastavujte u žádného připojení vyšší hodnotu než NUMCPUVPS.

c_per_t
počet připojení na jeden vyzvaný jednotkový proces. Toto číslo vypočtete podle následujícího vzorce:

```
c_per_t = connections / poll_threads
```

connections
je maximální předpokládaný počet připojení, který bude označený typ připojení podporovat. U připojení prostřednictvím sdílené paměti (**ipcshm**) zadejte pro nejlepší výkon dvojnásobek počtu připojení.

Toto pole se používá pouze pro připojení prostřednictvím sdílené paměti v systému Windows. Jiné metody připojení v systému Windows tuto hodnotu ignorují.

vp_class
je třída virtuálního procesoru, kde lze spouštět vyzvané jednotkové procesy. Zadejte CPU, máte-li jediný vyzvaný jednotkový proces běžící ve virtuálním procesoru CPU. Potřebujete-li více vyzvaných jednotkových procesů, zadejte pro nejlepší výkon parametr NET. Používáte-li systém Windows, zadejte parametr NET v každém případě. Výchozí hodnota tohoto pole závisí na následujících podmínkách:

- Je-li typ připojení přidružený k názvu serveru, který je uveden v parametru DBSERVERNAME, a pokud žádný předchozí parametr NETTYPE výslovně neurčuje CPU, je výchozí třídou virtuálního procesoru CPU. Je-li třída CPU již obsazena, je výchozí třídou NET.
- Pokud je typ připojení přidružený k názvu serveru, který je uveden v parametru DBSERVERALIASES, je výchozí třídou virtuálního procesoru NET.

Pokud hodnota *c_per_t* přesáhne 350 a počet vyzvaných jednotkových procesů pro aktuální připojení je menší než NUMCPUVPS, můžete zlepšit výkon určením třídy NET CPU, přidáním vyzvaných jednotkových procesů (nepřekračujte hodnotu NUMCPUVPS) a novým výpočtem hodnoty *c_per_t*. Výchozí hodnota *c_per_t* je 50.

Důležité: Každé připojení **ipcshm** vyžaduje semafor. Některé operační systémy vyžadují, abyste nakonfigurovali maximální počet semaforů, které mohou být vyžadovány všemi soubory programů spuštěnými v počítači. Chcete-li dosáhnout nejlepšího výkonu, pak při přidělování semaforů pro komunikaci prostřednictvím sdílené

paměti vynásobte aktuální počet připojení **ipcshm** dvěma. Další informace naleznete v části “Parametry semaforu systému UNIX” na stránce 3-2.

Máte-li jednoprocesorový počítač a instance databázového serveru je nakonfigurována pouze pro jeden typ připojení, můžete vynechat parametr NETTYPE. Databázový server používá informace v souboru **sqlhosts** nebo v registru k navázání připojení typu klient/server.

Pokud je počítač jednoprocesorový a instance databázového serveru je nakonfigurována pro více než jeden typ připojení, vložte samostatný parametr NETTYPE pro každý typ připojení. Pokud počet připojení libovolného typu značně převyšuje hodnotu 300, přiřaďte dva nebo více vyzvaných jednotkových procesů (maximálně do hodnoty NUMCPUVPS) a určete třídu virtuálního procesoru NET podle následujícího příkladu:

```
NETTYPE ipcshm,1,200,CPU
NETTYPE tlitcp,2,200,NET # podporuje 400 připojení
```

V případě ipcshm odpovídá počet vyzvaných jednotkových procesů počtu segmentů v paměti. Pokud je například parametr NETTYPE nastaven na hodnotu 3,100 a vytváří pouze jeden jednotkový proces, nastavte jej na hodnotu 1,300. Chcete-li jeden vyzvaný jednotkový proces, nastavte vyzvaný jednotkový proces na hodnotu 1,300.

Máte-li víceprocesorový počítač, instance databázového serveru je nakonfigurována pouze pro jeden typ připojení a počet připojení nepřekračuje hodnotu 350, můžete pomocí parametru NETTYPE určit jeden vyzvaný jednotkový proces ve virtuálním procesoru třídy CPU nebo NET. Přesahuje-li počet připojení hodnotu 350, nastavte třídu VP na NET, zvyšte počet vyzvaných jednotkových procesů a znovu vypočítejte hodnotu *c_per_t*.

Poznámka: Měli byste pečlivě rozlišovat mezi vyzvanými jednotkovými procesy pro síťová připojení a vyzvanými jednotkovými procesy pro připojení prostřednictvím sdílené paměti, které by měly být spuštěny vždy jeden v každém virtuálním procesoru CPU. Připojení TCP by měla být vytvářena pouze v síťových virtuálních procesorech, a to v minimálním počtu potřebném k dosažení požadované odezvy. Připojení prostřednictvím sdílené paměti by měla být vytvořena pouze ve virtuálních procesorech CPU a měla by být spuštěna v každém virtuálním procesoru CPU.

Povolení rychlého dotazování pomocí konfiguračního parametru FASTPOLL

Pomocí konfiguračního parametru FASTPOLL můžete v síti povolit nebo zakázat rychlé dotazování (pokud operační systém rychlé dotazování podporuje). Rychlé dotazování je užitečné v případě, že máte velký počet připojení. Máte-li například více než 300 souběžných připojení k databázovému serveru, můžete povolením konfiguračního parametru FASTPOLL zlepšit výkon. Další informace naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server* a v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Společné oblasti vyrovnávacích pamětí v síti

Velikost vyrovnávacích pamětí pro připojení TCP/IP ovlivňuje využití paměti a CPU. Nastavením těchto vyrovnávacích pamětí tak, aby zvládly obvyklé požadavky, může zlepšit využití CPU odstraněním nutnosti rozdělovat požadavky do více zpráv. Tuto schopnost je nicméně třeba používat opatrně. Databázový server dynamicky přiděluje vyrovnávací paměti zadané velikosti aktivním připojením. Pokud velikost vyrovnávacích pamětí nenastavíte opatrně, mohou spotřebovat mnoho paměti. Podrobnosti o nastavení velikosti vyrovnávací paměti sítě naleznete v části “Nastavení velikosti vyrovnávacích pamětí pomocí proměnné prostředí IFX_NETBUF_SIZE” na stránce 3-17.

Databázový server dynamicky přiděluje požadavkům klientů síťové vyrovnávací paměti z globální společné oblasti. Po zpracování požadavků klientů vrátí databázový server paměť do společné oblasti vyrovnávacích pamětí v síti, která je sdílená relacemi, které používají připojení SOCTCP, IPCSTR nebo TLITCP.

Tato společná síťová oblast vyrovnávacích pamětí má následující výhody:

- Zabraňuje častému přidělování a následnému zrušení přidělení z globálního společné oblasti paměti.
- Používá méně prostředků CPU k přidělení a rušení přidělení vyrovnávacích pamětí sítě ze společného prostoru vyrovnávacích pamětí v síti pro každý přenos v síti.
- Snižuje soupeření o přidělování sdílené paměti.

Volná společná oblast vyrovnávací paměti sítě se může během období špiček aktivity zvětšit. Pro zabránění stavu, kdy v těchto společných oblastech vyrovnávací paměti sítě zůstalo velké množství nevyužité paměti v době, kdy již není aktivita sítě vysoká, vrací databázový server volné vyrovnávací paměti zpět ve chvíli, kdy počet volných vyrovnávacích pamětí dosáhne určité prahové hodnoty.

Databázový server poskytuje následující funkce pro další snížení frekvence přidělování vyrovnávacích pamětí a snížení soupeření o volné vyrovnávací paměti v síti:

- Soukromá společná oblast volné vyrovnávací paměti sítě pro zabránění častého přidělování vyrovnávacích pamětí sítě ze společného prostoru vyrovnávací paměti sítě nebo z globálního společné paměťové oblasti ve sdílené paměti.
- Schopnost určit pro příjem síťových paketů nebo zpráv od klientů velikost vyrovnávací paměti větší, než 4 kilobajty.

Jako administrátor systému můžete prahové hodnoty volné vyrovnávací paměti a velikost každé vyrovnávací paměti nastavit pomocí jedné z následujících možností:

- Konfigurační parametr NETTYPE
- Proměnná prostředí **IFX_NETBUF_PVTPOOL_SIZE**
- Proměnná prostředí **IFX_NETBUF_SIZE** a volba **b** (velikost vyrovnávací paměti klienta) v souboru **sqlhosts** nebo v registru

Použití konfiguračního parametru NETTYPE ve vztahu ke společným oblastem vyrovnávací paměti sítě

Databázový server implementuje prahovou hodnotu volných vyrovnávacích pamětí sítě za účelem zabránění častému přidělování a rušení přidělení sdílené paměti pro společnou oblast vyrovnávacích pamětí sítě. Tato prahová hodnota umožňuje databázovému serveru udržet takový počet volných vyrovnávacích pamětí sítě, který odpovídá počtu připojení, zadanému konfiguračním parametrem NETTYPE.

Databázový server dynamicky přiděluje vyrovnávací paměti sítě pro zprávy s požadavky klientů. Po zpracování požadavků klientů serverem jsou vyrovnávací paměti vráceny do společné oblasti volných vyrovnávacích pamětí sítě.

Je-li počet volných vyrovnávacích pamětí větší než prahová hodnota, databázový server vrátí paměť přidělenou vyrovnávacím pamětem, které přesahují prahovou hodnotu, do globální společné oblasti.

Databázový server počítá prahovou hodnotu volných vyrovnávacích pamětí ve společném prostoru vyrovnávacích pamětí sítě podle následujícího vzorce:

```
prahová hodnota =
   100 + (0.7 * number_connections)
```

Hodnota *number_connections* je celkový počet připojení zadaný ve třetím poli položky NETTYPE pro různé typy síťových připojení (SOCTCP, IPCSTR nebo TLITCP). V tomto vzorci není použita hodnota položky NETTYPE pro sdílenou paměť (IPCSHM).

Nezadáte-li hodnotu ve třetím poli parametru NETTYPE, databázový server použije výchozí hodnotu 50 připojení pro každou položku NETTYPE, která odpovídá protokolům SOCTCP, TLITCP a IPCSTR.

Povolení podpory soukromých vyrovnávacích pamětí sítě pomocí proměnné prostředí IFX_NETBUF_PVTPOOL_SIZE

Databázový server podporuje soukromé vyrovnávací paměti sítě pro každou relaci, která používá síťová připojení SOCTCP, IPCSTR nebo TLITCP.

V situacích, kdy je neustále aktivních mnoho připojení a relací, mají tyto soukromé vyrovnávací paměti následující výhody:

- Méně soupeření o obecnou společnou oblast vyrovnávacích pamětí sítě
- Méně prostředků CPU k přidělení a rušení přidělení vyrovnávacích pamětí sítě z obecného společného prostoru vyrovnávacích pamětí sítě pro každý přenos v síti

Proměnná prostředí **IFX_NETBUF_PVTPOOL_SIZE** určuje velikost soukromé společné oblasti vyrovnávacích pamětí sítě pro každou relaci. Výchozí velikost je jedna vyrovnávací paměť. Další informace o této proměnné prostředí naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Pomocí následujících voleb programu **onstat** můžete monitorovat využití vyrovnávacích pamětí sítě.

Volba	Pole výstupu	Popis
onstat -g ntu	q-pvt	Aktuální počet a nejvyšší počet vyrovnávacích pamětí, které jsou pro tuto relaci volné v soukromé společné oblasti.
onstat -g ntm	q-exceeds	Počet překročení prahové hodnoty počtu volných vyrovnávacích pamětí.

Volba **onstat -g ntu** zobrazuje následující formát pole výstupu **q-pvt**:

```
aktuální počet / nejvyšší počet
```

Je-li počet volných vyrovnávacích pamětí (hodnota v poli **q-pvt**) trvale 0, můžete provést jednu z následujících akcí:

- Zvýšit počet vyrovnávacích pamětí pomocí proměnné prostředí **IFX_NETBUF_PVTPOOL_SIZE**.
- Zvětšit jednotlivé vyrovnávací paměti pomocí proměnné prostředí **IFX_NETBUF_SIZE**.

Pole **q-exceeds** označuje, kolikrát byla překročena prahová hodnota společné oblasti volných vyrovnávacích pamětí sítě. Po překročení této prahové hodnoty databázový server vrací nepoužívané vyrovnávací paměti sítě (ty, které přesahují tuto prahovou hodnotu) do globální společné oblasti paměti ve sdílené paměti. V optimálním případě by měla být tato hodnota 0, případně nízká hodnota, což znamená že server nepřiděluje nebo neruší přidělení vyrovnávacích pamětí sítě z globální společné oblasti paměti.

Nastavení velikosti vyrovnávacích pamětí pomocí proměnné prostředí IFX_NETBUF_SIZE

Proměnná prostředí **IFX_NETBUF_SIZE** určuje velikost jednotlivých vyrovnávacích pamětí v obecné společné oblasti vyrovnávacích pamětí sítě a v soukromé společné oblasti vyrovnávacích pamětí sítě. Výchozí velikost vyrovnávací paměti je 4 kilobajty.

Proměnná prostředí **IFX_NETBUF_SIZE** umožňuje databázovému serveru během jednoho volání systému přijímat zprávy větší než 4 kilobajty. Větší vyrovnávací paměť snižuje velikost režie, která je nutná pro příjem každého paketu.

Zvyšte hodnotu IFX_NETBUF_SIZE v případě, že chcete, aby klienti odesílali větší pakety než 4 kilobajty. Klienti odesílají velké pakety v těchto situacích:

- Replikace databázové tabulky
- Vložení řádků větších než 4 kilobajty
- Odesílání inteligentních velkých objektů

Volba **b** v souboru **sqlhosts** umožňuje klientovi odesílat a přijímat více více než 4 kilobajty. Hodnota volby **sqlhosts** by měla obvykle odpovídat hodnotě IFX_NETBUF_SIZE. Další informace o volbách v souboru **sqlhosts** naleznete v příručce *IBM Informix Administrator's Guide*.

Pomocí následujícího příkazu obslužného programu **onstat** lze zobrazit velikost vyrovnávací paměti sítě:

```
onstat -g afr global | grep net
```

Pole **size** ve výstupu zobrazuje velikost vyrovnávací paměti sítě v bajtech.

Virtuální procesory a využití CPU

Pokud je databázový server v režimu online, umožňuje spouštět a zastavovat virtuální procesory, které náleží do určitých tříd. V době, kdy je databázový server v režimu online, můžete pomocí obslužných programů **onmode -p** nebo IBM Informix Server Administrator (ISA) nebo ON–Monitor spustit další virtuální procesory z následujících tříd: CPU, AIO, PIO, LIO, SHM, TLI a SOC. Virtuální procesory třídy CPU můžete zastavit pouze v případě, že je databázový server v režimu online.

Měli byste pečlivě rozlišovat mezi vyzvanými jednotkovými procesy pro síťová připojení a vyzvanými jednotkovými procesy pro připojení prostřednictvím sdílené paměti, které by měly být spuštěny vždy jeden v každém virtuálním procesoru CPU. Připojení TCP by měla být vytvářena pouze v síťových virtuálních procesorech, a to v minimálním počtu potřebném k dosažení požadované odezvy. Připojení prostřednictvím sdílené paměti by měla být vytvořena pouze ve virtuálních procesorech CPU a měla by být spuštěna v každém virtuálním procesoru CPU.

Přidání virtuálních procesorů

Při každém přidání síťového virtuálního procesoru (SOC nebo TLI) přidáte také vyzvaný jednotkový proces. Každý vyzvaný jednotkový proces je spuštěn v odděleném virtuálním procesoru CPU nebo v síťovém virtuálním procesoru sítě příslušného typu. Přidání dalších virtuálních procesorů může zvýšit zátěž zdrojů CPU. Pokud tedy hodnota NETTYPE ukazuje, že dostupný virtuální procesor CPU může obsloužit daný vyzvaný jednotkový proces, databázový server přiřadí tento proces k tomuto virtuálnímu procesoru CPU. Mají-li všechny virtuální procesory CPU přiděleny vyzvané jednotkové procesy, databázový server přidá druhý síťový virtuální procesor, sloužící k obsluze daného vyzvaného jednotkového procesu.

Monitorování virtuálních procesorů

Monitorování virtuálních procesorů slouží ke zjištění, zdali je počet virtuálních procesorů konfigurovaných pro databázový server optimální pro aktuální úroveň aktivity.

Sledování virtuálních procesorů:

- K zobrazování informací použijte obslužné programy příkazového řádku, například **onstat-g ioq**. Další informace naleznete v části "Použití obslužných programů příkazového řádku ke sledování virtuálních procesorů".
- Konfigurační parametr AUTO_AIOVPS umožňuje povolit databázovému serveru, aby zvětšil počet virtuálních procesorů AIO a jednotkových procesů čištění stránek, pokud server zjistí, že virtuální procesory nepostačují zátěži vstupu - výstupu.
- Dotazy na tabulky SMI. Další informace naleznete v části "Použití SMI ke sledování virtuálních procesorů" na stránce 3-20.

Použití obslužných programů příkazového řádku ke sledování virtuálních procesorů

K monitorování virtuálních procesorů můžete použít následující obslužné programy příkazového řádku.

onstat -g glo: Pomocí volby **onstat -g glo** zobrazíte informace o každém virtuálním procesoru, který je aktuálně spuštěn, a také souhrnné statistiky pro každou třídu virtuálních procesorů.

Pole **vps** ve výstupu zobrazuje aktuální počet aktivních virtuálních procesorů dané třídy. V příkladu výstupu obslužného programu **onstat -g glo**, který zobrazuje Obrázek 3-1, je v poli **vps** zobrazeno, že jsou aktuálně aktivní 3 virtuální procesory CPU.

Použitím volby **onstat -g rea** podle popisu v části "onstat -g rea" na stránce 3-19 určíte, zda je nutné zvýšit počet virtuálních procesorů.

```
MT global info:
sessions threads  vps      lngspins
1        15       8        0

Virtual processor summary:
class    vps      usercpu          syscpu           total
 cpu       3       479.77          190.42          670.18
 aio       1         0.83            0.23            1.07
 pio       1         0.42            0.10            0.52
 lio       1         0.27            0.22            0.48
 soc       0         0.00            0.00            0.00
 tli       0         0.00            0.00            0.00
 shm       0         0.00            0.00            0.00
 adm       1         0.10            0.45            0.55
 opt       0         0.00            0.00            0.00
 msc       1         0.28            0.52            0.80
 adt       0         0.00            0.00            0.00
 total     8       481.67          191.93          673.60

Individual virtual processors:
   vp        pid         class    usercpu          syscpu           total
    1        1776         cpu      165.18           40.50          205.68
    2        1777         adm        0.10            0.45            0.55
    3        1778         cpu      157.83           98.68          256.52
    4        1779         cpu      156.75           51.23          207.98
    5        1780         lio        0.27            0.22            0.48
    6        1781         pio        0.42            0.10            0.52
    7        1782         aio        0.83            0.23            1.07
    8        1783         msc        0.28            0.52            0.80
                          tot      481.67          191.93          673.60
```

Obrázek 3-1. Výstup onstat -g glo

onstat -g rea: K monitorování počtu jednotkových procesů ve frontě připravených procesů použijte volbu **onstat -g rea**. V poli **status** ve výstupu je zobrazena hodnota ready v případě, že je daný jednotkový proces ve frontě připravených procesů. Pole **vp-class** ve výstupu zobrazuje třídu virtuálního procesoru, ve které je daný jednotkový proces proveden. Roste-li počet jednotkových procesů ve frontě připravených procesů pro určitou třídu virtuálních procesorů (například třída CPU), bude pravděpodobně nutné do konfigurace přidat více těchto virtuálních procesorů. Obrázek 3-2 zobrazuje vzorový výstup obslužného programu **onstat -g rea**.

```
Ready threads:
tid     tcb      rstcb   prty    status                vp-class   name

6       536a38   406464  4       ready                   3cpu    main_loop()
28      60cfe8   40a124  4       ready                   1cpu    onmode_mon
33      672a20   409dc4  2       ready                   3cpu    sqlexec
```

Obrázek 3-2. Výstup onstat -g rea

onstat -g ioq: Pomocí volby **onstat -g ioq** lze zjistit, zda je nutné přidělit další virtuální procesory typu AIO. Příkaz **onstat -g ioq** zobrazuje délku front vstupu - výstupu ve sloupci **len**, jak ukazuje Obrázek 3-3. Dále můžete ve sloupci **maxlen** vidět maximální délku fronty (od chvíle spuštění databázového serveru). Pokud délka fronty vstupu - výstupu narůstá, hromadí se požadavky vstupu - výstupu rychleji, než mohou být zpracovány virtuálními procesory typu AIO. Naznačuje-li délka fronty vstupu - výstupu, že dochází k hromadění požadavků vstupu - výstupu, zvažte možnost přidání virtuálních procesorů typu AIO.

```
onstat -g ioq

AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
  adt   0      0      0        0        0        0        0
  msc   0      0      1       12        0        0        0
  aio   0      0      4       89       68        0        0
  pio   0      0      1        1        0        1        0
  lio   0      0      1       17        0       17        0
  kio   0      0      0        0        0        0        0
  gfd   3      0      3      254      242       12        0
  gfd   4      0     17      614      261      353        0


onstat -d
Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
a1de1d8  1        1        1        1        N        informix rootdbs
a1df550  2        1        2        1        N        informix space1
 2 active, 32,678 maximum
Chunks
address  chk/dbs offset   size     free    bpages   flags pathname
a1de320  1   1   0        75000    66447             PO-   /ix/root_chunk
a1df698  2   2   0        500      447               PO-   /ix//chunk1
 2 active, 32,678 maximum
```

Obrázek 3-3. Výstupy onstat -g ioq a onstat -d

Každý blok, obsluhovaný virtuálním procesorem AIO, má ve výstupu příkazu **onstat -g ioq** jeden řádek, identifikovaný hodnotou gfd ve sloupci **q name**. Můžete sladit řádek ve výstupu příkazu **onstat -g ioq** se skutečným blokem, protože bloky jsou ve stejném pořadí jako ve výstupu příkazu **onstat -d**. Ve výstupu příkazu **onstat -g ioq** (Obrázek 3-3) jsou dvě fronty gfd. První fronta gfd obsahuje požadavky bloku **root_chunk**, protože odpovídá prvnímu bloku zobrazenému ve výstupu příkazu **onstat -d** (Obrázek 3-3). Podobně, druhá fronta gfd obsahuje požadavky bloku **chunk1**, protože odpovídá druhému bloku ve výstupu příkazu **onstat -d**.

Má-li databázový server k dispozici směs přímých zařízení a předpřipravených souborů, fronty gfd ve výstupu příkazu **onstat -d** odpovídají pouze předpřipraveným souborům.

Použití ISA ke sledování virtuálních procesorů

Chcete-li monitorovat virtuální procesory pomocí programu ISA, klepněte na stránku **VPs** na hlavní stránce programu ISA. Program ISA používá informace vygenerované volbou příkazového řádku **onstat -g glo** . Klepnutím na tlačítko **Refresh** spustíte příkazy znovu a zobrazíte aktuální informace.

Použití SMI ke sledování virtuálních procesorů

Chcete-li provést dotaz v tabulkách SMI, je nutné se připojit k databázi **sysmaster**. Informace o právě spuštěných virtuálních procesorech vyhledáte v tabulce SMI **sysvpprof**. Tato tabulka obsahuje následující sloupce.

Sloupec	Popis
vpid	Číslo ID virtuálního procesoru.
class	Třída virtuálního procesoru.
usercpu	Spotřebované sekundy času uživatelské jednotky CPU.
syscpu	Spotřebované sekundy času systémové jednotky CPU.

Mezipaměti virtuálních procesorů CPU

Každý virtuální procesor (VP) CPU může mít volitelně přiřazenou privátní mezipaměť. Každý blok soukromé mezipaměti se skládá z 1 až 32 stránek paměti, kde každá paměťová stránka má 4096 bajtů. Účelem této mezipaměti je zlepšit přístupovou dobu k blokům paměti.

Za normálních okolností databázový server při hledání paměťových bloků požadované délky prohledává bitovou mapu. Za určitých okolností může být toto hledání pomalé. Při tomto hledání je také nutné provést uzamknutí, což může ovlivnit souběžné procesy u velkých víceprocesorových počítačů. Použitím soukromé mezipaměti se lze tomuto problému vyhnout a zlepšit souběžnost.

Soukromá mezipaměť může zvýšit spotřebu paměti.

Konfigurační parametr VP_MEMORY_CACHE_KB umožňuje povolit soukromou mezipaměť a zadat informace o mezipaměti. Další informace naleznete v části "Povolení soukromých mezipamětí pomocí konfiguračního parametru VP_MEMORY_CACHE_KB". Informace naleznete také v části "Získávání statistik mezipaměti paměťových bloků virtuálních procesorů CPU" na stránce 3-22 a informace o konfiguračním parametru VP_MEMORY_CACHE_KB naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Povolení soukromých mezipamětí pomocí konfiguračního parametru VP_MEMORY_CACHE_KB

Konfigurační parametr VP_MEMORY_CACHE_KB umožňuje povolit soukromou mezipaměť a zadat informace o mezipaměti.

Povolení soukromé mezipaměti:

1. Zjistěte hodnotu zadanou v konfiguračním parametru SHMTOTAL. Tato hodnota určuje celkovou velikost sdílené paměti (rezidentní, virtuální, komunikační a části virtuálních přípon), kterou databázový server používá pro všechna přidělení paměti.
2. Nastavte konfigurační parametr VP_MEMORY_CACHE_KB, který určuje velikost paměti pro každý virtuální procesor CPU, na hodnotu rozsahu od minimálně 800 kilobajtů do velikosti, která nepřesahuje 40 procent omezení paměti, které je určeno v konfiguračním parametru SHMTOTAL.

Ve většině případů doporučujeme nastavit konfigurační parametr VP_MEMORY_CACHE_KB na hodnotu výrazně menší, než je maximum 40 procent omezení paměti, určené konifugračním parametrem SHMTOTAL. Menší hodnota by měla vést k vysokému poměru přístupu do mezipaměti a také nedojde k vázání paměťových bloků na jediný virtuální procesor CPU. Dále, zatímco normální rutiny databázového serveru pro správu paměti mohou kombinovat sousední volné paměťové bloky, mezipaměť virtuálního procesoru CPU sousední volné paměťové bloky nekombinuje.

Chcete-li změnit údaje mezipaměti, použijte volby **onmode -wf** nebo **-wm**. Volba **onmode -wf** provede změnu údajů okamžitě.

Nastavíte-li tyto volby **onmode** na hodnotu 0, databázový server vyprázdní a zakáže soukromé mezipaměti.

Další informace naleznete v části "Získávání statistik mezipaměti paměťových bloků virtuálních procesorů CPU" na stránce 3-22 a podrobné informace o konfiguračních parametrech naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Získávání statistik mezipaměti paměťových bloků virtuálních procesorů CPU

Chcete-li zobrazit statistiky paměťových bloků virtuálních procesorů CPU, použijte volbu **onstat -g vpcache**.

Připojení a využití CPU

Některé aplikace mají velký počet připojení typu klient/server. Otevírání a zavírání připojení může spotřebovat hodně času systémového CPU. Následující části popisují způsoby, jimiž se pravděpodobně podaří snížit čas systémového CPU potřebný k otevírání a zavírání připojení.

Multiplexní připojení

Mnoho tradičních klientských aplikací SQL, které nejsou spuštěny v podobě jednotkových procesů, používá více připojení k databázi k tomu, aby bylo možné vykonat práci pro jediného uživatele. Každé připojení k databázi naváže oddělené síťové připojení k databázovému serveru.

Možnost multiplexního připojení pro databázový server poskytuje schopnost obsluhovat jedním síťovým připojením v databázovém serveru více připojení k databázi z klientské aplikace do tohoto databázového serveru.

Pokud klient, který neběží v jednotkovém procesu, použije multiplexní připojení, databázový server přesto vytvoří stejný počet uživatelských relací a uživatelských podprocesů jako v případě, kdy nejde o multiplexní připojení. Počet síťových připojení se nicméně při použití multiplexních připojení sníží. Databázový server namísto toho použije jednotkový proces multiplexního modulu listener, což umožní více připojením k databázi sdílet stejné síťové připojení.

Chcete-li zlepšit dobu odezvy u klientů, kteří neběží v jednotkovém procesu, můžete pro provádění dotazů SQL použít multiplexní připojení. Rozsah zlepšení výkonu závisí na následujících faktorech:

- Snížení celkového počtu síťových připojení a výsledné snížení času systémového CPU.

  Obvyklou příčinou velkého množství času systémového CPU je zpracování systémových volání pro síťové připojení. Z tohoto důvodu největší snížení času systémového CPU odpovídá snížení celkového počtu síťových připojení.
- Poměr tohoto snížení času systémového CPU k času uživatelského CPU.

  Jsou-li dotazy jednoduché a využijí jen málo času CPU, je možné, že dojde při použití multiplexního připojení ke značnému snížení doby odezvy. Jsou-li ovšem dotazy rozsáhlé a využívají hodně času uživatelského CPU, je možné, že nedojde ke zlepšení výkonu.

  Chcete-li získat představu o množství času systémového CPU uživatelského CPU na virtuální procesor, použijte volbu **onstat -g glo**.

Chcete-li použít multiplexní připojení pro klientskou aplikaci, která není spuštěna v jednotkovém procesu, je třeba před spuštěním databázového serveru provést následující kroky:

1. Definovat alias pomocí konfiguračního parametru DBSERVERALIAS. Zadejte například:

   ```
   DBSERVERALIAS ids_mux
   ```
2. Přidejte položku souboru SQLHOSTS pro tento alias pomocí příkazu sqlmux jako položku **nettype**, což je druhý sloupec v souboru SQLHOSTS. Zadejte například:

   ```
   ids_mux  onsqlmux  ......
   ```

   Další pole této položky, **hostname** a **servicename**, musejí být přítomna, ale jsou ignorována.

3. Povolit multiplexní funkci pro dané připojení zadáním hodnoty m=1 do souboru **sqlhosts** nebo do registru, který klient používá pro připojení k databázovému serveru.
4. Na platformách Windows musíte zadat také proměnnou prostředí **IFX_SESSION_MUX**.

Upozornění: V systému Windows nesmí aplikace s vícenásobnými jednotkovými procesy používat funkci multiplexního připojení. Má-li aplikace s vícenásobnými jednotkovými procesy povolenou multiplexní funkci v souboru **sqlhosts** nebo v položce registru a je-li současně definovaná proměnná prostředí IFX_SESSION_MUX, může to vést ke katastrofálním následkům, včetně havárií a poškození dat.

Další informace o omezeních multiplexních připojení naleznete v příručkách *IBM Informix ESQL/C Programmer's Manual* a *Příručka administrátora serveru IBM Informix Dynamic Server*.

Program MaxConnect pro vícenásobná připojení (UNIX)

Program IBM Informix MaxConnect je síťový produkt pro prostředí databázového serveru Informix v systému UNIX. Program MaxConnect může spravovat velké množství (od několika set do desítek tisíc) spojení typu klient-server.

Program MaxConnect je nejvhodnější pro přenosy dat OLTP a není doporučeno ho používat ho k přenosům velkých objemů multimediálních dat. Program MaxConnect poskytuje pro střední až velké konfigurace OLTP následující výhody co se týká výkonu:

- Snižuje požadavky na CPU databázového serveru snížením počtu fyzických připojení.

  Program MaxConnect používá multiplexní připojení, takže poměr mezi klientskými připojeními a připojeními k databázi může být i 100:1 a vyšší.
- Zlepšuje dobu odezvy pro koncového uživatele tím, že zvyšuje přizpůsobitelnost systému na mnoho tisíc připojení.
- Snižuje zahlcení operačního systému agregací mnoha malých paketů do jedné přenosové operace.

Abyste dosáhli maximálního zvýšení výkonu, nainstalujte program MaxConnect na jeden z vyhrazených počítačů, ke kterému se připojují klienti systému Informix, nebo na klienstký aplikační server. Každá z těchto konfigurací snižuje požadavky na CPU na vyřízení velkého počtu připojení v počítači databázového serveru.

Program MaxConnect můžete monitorovat pomocí příkazu **onstat -g imc** v počítači databázového serveru a pomocí příkazu **imcadmin** v počítači, kde se nachází program MaxConnect.

Další informace o instalaci, konfiguraci, monitorování a ladění MaxConnect naleznete v příručce *IBM Informix MaxConnect User's Guide*.

Důležité: Program MaxConnect a příručka *IBM Informix MaxConnect User's Guide* nejsou součástí dodávky produktu IBM Informix Dynamic Server.

Kapitola 4. Vliv konfigurace na využití paměti

Obsah kapitoly

Tato kapitola pojednává o kombinovaném vlivu konfiguračních parametrů operačního systému a databázového serveru na využití paměti. Tato kapitola pojednává o parametrech, které přímo ovlivňují využití paměti, a vysvětluje způsob jejich nastavení. Tam, kde je to možné, tato kapitola také poskytne doporučená nastavení nebo možnosti, které lze použít při jiných pracovních využitích.

Při přidělování sdílené paměti databázového serveru berte v úvahu velikost fyzické paměti dostupné v hostitelském počítači. Obecně platí, že zvýšením velikosti sdílené paměti databázového serveru zvýšíte jeho výkonnost. Je třeba vyvážit velikost sdílené paměti přidělené databázovému serveru a velikost paměti potřebné pro provoz virtuálních procesorů a dalších procesů.

Přidělování sdílené paměti

V operačním systému je třeba pro databázový server nakonfigurovat adekvátní prostředky sdílené paměti. Nedostatek sdílené paměti může nepříznivě ovlivnit výkon. Když operační systém přidělí blok sdílené paměti, je tento blok nazýván *segmentem*. Když databázový server připojí celý segment sdílené paměti nebo jeho část, je tato paměť nazývána *částí*.

Databázový server používá následující části sdílené paměti. Každá část tvoří samostatný díl z celkového množství sdílené paměti, které databázový server vyžaduje:

- Rezidentní část
- Virtuální část
- Část zpráv

Rezidentní část a část zpráv jsou statické a je pro ně tedy třeba před uvedením databázového serveru do režimu online přidělit dostatek paměti. (Obvykle je pro přednastavení sdílené paměti nutné restartovat operační systém.) Virtuální část sdílené paměti databázového serveru se sice rozšiřuje dynamicky, ale stále je nutné při přidělování systémové sdílené paměti pro tuto část zajistit dostatečnou počáteční velikost.

Následující části poskytují pokyny k odhadnutí velikosti každé části sdílené paměti databázového serveru, abyste mohli v operačním systému přiřadit adekvátní velikost paměti. Požadovaná velikost je součet potřebných velikostí všech tří částí, které určuje konfigurační parametr SHMTOTAL.

Rezidentní část

Rezidentní část obsahuje oblasti sdílené paměti, které zaznamenávají stav databázového serveru včetně vyrovnávacích pamětí, uzamčení, soubory protokolů a umístění dbspaces, chunks a tblspaces.

Nastavení, která používáte pro konfigurační parametry LOCKS, LOGBUFF a PHYSBUFF pomohou stanovit velikost rezidentní části.

Konfigurační parametr BUFFERPOOL stanovuje množství vyrovnávacích pamětí přiřazených k rezidentnímu segmentu při spuštění databázového serveru. následující vyrovnávací paměti, přidané do databázového serveru za chodu, jsou až do restartu databázového serveru uloženy ve virtuální paměti.

Navíc k těmto parametrům, které ovlivňují velikost rezidentní části, konfigurační parametr RESIDENT ovlivňuje využití paměti. Je-li parametr RESIDENT v souboru ONCONFIG v počítači, který podporuje vynucené uložení, nastaven na hodnotu 1, není rezidentní část nikdy stránkována.

Soubor Poznámky k počítači ukazuje vašemu databázovému serveru, zda váš operační systém podporuje vynucené uložení. Informace o umístění souboru Poznámky k počítači naleznete v úvodu této příručky.

Chcete-li při přiřazování sdílené paměti operačního systému odhadnout velikost rezidentní části (v jednotkách kilobajtů), postupujte následovně. Výsledek je hodnota lehce převyšující množství paměti, které rezidentní část skutečně využije.

Odhadnutí velikosti rezidentní části:

1. Chcete-li odhadnout velikost vyrovnávací paměti dat, použijte následující vzorec:

   ```
   velikost vyrovnávací paměti = (BUFFERS * velikost stránky) + (BUFFERS * 254)
   ```

   velikost stránky je velikost stránky sdílené paměti, obslužný program onstat spuštěný s volbou b (**onstat -b**) ji zobrazuje na posledním řádku v poli **velikost vyrovnávací paměti**.

2. Vypočítejte hodnoty následujících vzorců:

   ```
   hodnota locks = LOCKS * 44
   hodnota logbuff = LOGBUFF * 1024 * 3
   hodnota physbuff = PHYSBUFF * 1024 * 2
   ```

3. Chcete-li vypočítat odhadovanou velikost rezidentní části v kilobajtech, použijte následující vzorec:

   ```
   rsegsize = (velikost vyrovnávací paměti + hodnota locks +  hodnota logbuff
              + hodnota physbuff + 51,200) / 1024
   ```

Tip: Počáteční velikost tabulky zámků určuje konfigurační parametr LOCKS. Pokud počet zámků přidělených relacemi překročí hodnotu parametru LOCKS, databázový server dynamicky zvětší velikost tabulky zámků. Pokud předpodkládáte, že tabulka zámku bude dynamicky zvětšována, nastavte parametr SHMTOTAL na hodnotu *0*. Když je hodnota parametru SHMTOTAL *0*, není určen žádný limit na celkovou velikost paměti (virtuální rozšíření, rezidentní, virtuální a komunikační části sdílené paměti).

Další informace o konfiguračních parametrech BUFFERPOOL, LOCKS, LOGBUFF a PHYSBUFF naleznete v části “Konfigurační parametry, které ovlivňují využití paměti” na stránce 4-6.

Virtuální část

Virtuální část sdílené paměti obsahuje následující komponenty:

- Velké vyrovnávací paměti, které jsou používány ke čtení a zápisu velkých objemů dat při operacích Input/Output (I/O)
- Společné oblasti prostoru pro řazení
- Aktivní řídicí bloky jednotkových procesů, zásobníků a hald
- Data uživatelské relace
- Mezipaměti sloužící k uložení příkazů SQL, údajů datových slovníků a uživatelských rutin.
- Globální společná oblast pro vyrovnávací paměti síťového rozhraní a další údaje

Konfigurační parametr SHMVIRTSIZE v konfiguračním souboru databázového serveru poskytuje počáteční velikost virtuální části. Je-li třeba další místo ve virtuální části, bude databázový server zvyšovat velikost přidělené sdílené paměti po přírůstcích o velikosti dané konfiguračními parametry SHMADD nebo EXTSHMADD, a to až do omezení celkové přidělené sdílené paměti určeného parametrem SHMTOTAL.

Velikost virtuální části závisí zejména na typu aplikací a dotazů, které používáte. Počáteční odhad velikosti virtuální části může být v závislosti na vaší aplikaci pouze 100 kB na uživatele, ale také i 500 kB na uživatele. Navíc, chcete-li použít distribuce dat, je třeba k velikosti virtuální části přičíst další čtyři megabajty. Další informace o vytváření distribucí dat naleznete v pojednání o příkazu UPDATE STATISTICS v části “Vytvoření distribucí dat” na stránce 13-9.

Základní algoritmus pro stanovení odhadu počáteční velikosti virtuální části sdílené paměti je následující:

```
hodnota shmvirtsize = fixed overhead + sdílené paměťové struktury +
           (mncs * vlastní paměťové struktury) +
           další vyrovnávací paměti
```

Odhad hodnoty parametru SHMVIRTSIZE výše uvedeným vzorcem:

1. Hodnotu fixed overhead určíte podle následujícího vzorce:

   ```
   fixed overhead = globální společná oblast +
                    společná oblast jednotkového procesu po zavedení systému
   ```

   Velikosti společných oblastí přidělených relacím zjistíte použitím příkazu **onstat -g mem**. Počet bajtů přidělených pro jednu relaci vypočítáte odečtením hodnoty v poli **freesize** od hodnoty v poli **totalsize**.

   společná oblast jednotkového procesu po zavedení proměnné je částečně závislá na počtu virtuálních procesorů.

2. Hodnotu *sdílené paměťové struktury* odhadnete pomocí následujícího vzorce:

   ```
   shared structures = AIO vectors + sort memory +
               dbspace backup buffers +
               data-dictionary cache size +
               size of user-defined routine cache +
               histogram pool +
               STMT_CACHE_SIZE (mezipaměť příkazů SQL) +
               other pools (viz výstup příkazu onstat)
   ```

   Tabulka 4-1 zobrazuje umístění dalších informací o odhadování velikosti těchto sdílených struktur v paměti.

Tabulka 4-1. Informace o sdílených paměťových strukturách

Sdílená paměťová struktura	Podrobné informace
Paměť řazení	“Odhad paměti potřebné pro řazení” na stránce 7-13
vyrovnávací paměť datového slovníku	“Konfigurace datového slovníku” na stránce 4-20
Mezipaměť distribuce dat (společná oblast histogramu)	“Konfigurace distribuce dat” na stránce 4-22
Mezipaměť uživatelských rutin (UDR)	“Mezipaměť uživatelských rutin” na stránce 10-31
Mezipaměť příkazů jazyka SQL	“Povolení mezipaměti příkazů SQL” na stránce 13-29 “Mezipaměť příkazů SQL” na stránce 4-24
Jiné společné oblasti	Příkazem **onstat -g mem** zjistíte velikost paměti přiřazené různým společným oblastem.

3. Chcete-li předběžně vypočítat další část vzorce, postupujte takto:

   a. Pomocí následujícího vzroce odhadněte hodnotu *mncs* (což je maximální počet souběžných relací):

      ```
      mncs = number of poll threads *
             number connections per poll thread
      ```

      Do druhého pole konfiguračního parametru NETTYPE zadejte počet vyzvaných jednotkových procesů.

      Do třetího pole konfiguračního parametru NETTYPE zadejte *počet spojení na jeden vyzvaný jednotkový proces*.

      Hrubou hodnotu maximálního počtu souběžných relací můžete také získat zadáním příkazu **onstat -u** během vrcholného zatížení. Poslední řádek výstupu příkazu **onstat -u** obsahuje maximální počet souběžných uživatelských jednotkových procesů.

   b. Odhadněte hodnotu soukromých paměťových struktur pomocí následujícího vzorce:

      ```
      private structures = stack + heap +
                           session control-block structures
      ```

zásobník má obvykle velikost 32 bajtů, ale je závislý na rekurzi v uživatelských rutinách. Příkazem **onstat -g sts** lze velikost zásobníku zjistit.

halda má velikost zhruba 15 kB. Příkazem **onstat -g stm** lze velikost haldy pro příkazy SQL zjistit.

struktury řídicích bloků relace
vyjadřují množství paměti použité na jednu relaci. Příkaz **onstat -g ses** ve sloupci**total memory** zobrazí pro každé ID relace celkové množství paměti.

Více informací o příkazu **onstat -g stm** naleznete v části “Paměť relace” na stránce 4-35.

c. Následující část vzorce získáte vynásobením výsledků z kroků 3a a 3b:

```
mncs * private structures
```

4. Odhadněte hodnotu *other buffers* pro účet soukromých vyrovnávacích pamětí, kterým je přidělena paměť pro takové funkce, jako jsou odlehčené vstupně-výstupní operace inteligentních velkých objektů (přibližně 180 kB na uživatele).
5. Sečtením výsledků z kroků 1 až 4 získáte přibližnou hodnotu pro parametr SHMVIRTSIZE.

Tip: Běží-li databázový server pod stabilní zátěží, můžete k získání přesné hodnoty skutečné velikosti virtuální části použít příkaz **onstat -g seg**. Potom použijte tímto příkazem poskytnutou hodnotu shared memory k nastavení parametru SHMVIRTSIZE.

Část zprávy

Část zprávy obsahuje vyrovnávací paměť, kterou využívá komunikační rozhraní sdílené paměti. Množství potřebného prostoru pro tuto vyrovnávací paměť závisí na počtu uživatelských spojení, které pomocí daného síťového rozhraní povolíte. Pokud není určité rozhraní použito, nemusíte pro něj při přidělení paměti v operačním systému vyhrazovat místo. Pro odhad velikosti části paměti v kilobajtech můžete použít následující vzorec:

```
msegsize = (10,531 * ipcshm_conn + 50,000)/1024
```

ipcshm_conn je počet spojení, která lze provést přes rozhraní sdílené paměti, jak je definováno parametrem NETTYPE pro protokol **ipcshm**.

Konfigurace sdílené paměti UNIX

Následujícími kroky provedete konfiguraci segmentů sdílené paměti, která je vyžadována databázovým serverem. Více informací o nastavování parametrů souvisejících se sdílenou pamětí naleznete v pokynech k vašemu operačnímu systému.

Konfigurace segmentů sdílené paměti pro databázový server.:

1. Pokud nemá operační systém omezení velikosti segmentů sdílené paměti, postupujte následovně:
   a. Nastavte konfigurační parametr operačního systému pro maximální velikost segmentu, typicky parametry SHMMAX nebo SHMSIZE, na celkovou velikost, kterou vyžaduje databázový server. Tato velikost obsahuje množství paměti požadované pro spuštění instance databázového serveru a množství sdílené paměti přidělené pro dynamický růst virtuální části.
   b. Nastavte konfigurační parametr operačního systému pro maximální počet segmentů, typicky SHMMNI, alespoň na 1 pro každou instanci databázového serveru.
2. Pokud má operační systém omezení velikosti segmentu, postupujte následovně:

a. Nastavte konfigurační parametr operačního systému pro maximální velikost segmentu, typicky SHMMAX nebo SHMSIZE, na nejvyšší možnou hodnotu, kterou operační systém dovolí.

b. Pomocí následujícího vzorce vypočítejte počet segmentů pro vaši instanci databázového serveru. Pokud vyjde desetinné číslo, zaokrouhlete jej nahoru na nejbližší číslo celé.

```
SHMMNI = celková_sdílená_paměť / SHMMAX
```

celková_sdílená_paměť je celkové množství sdílené paměti, které vyhradíte pro použití databázovým serverem.

3. Nastavte konfigurační parametr operačního systému pro maximální počet segmentů, typicky SHMMNI, na hodnotu, která odpovídá celkovému množství sdílené paměti pro databázový server, vynásobené hodnotou SHMMAX nebo SHMSIZE. Pokud je počítač určen pro jednu instanci databázového serveru, může být tato hodnota až 90 procent velikosti virtuální paměti (fyzická paměť plus odkládací prostor).
4. Pokud operační systém používá ke zobrazení maximálního počtu segmentů sdílené paměti, kterou si může proces připojit, konfigurační parametr SHMSEG, nastavte jej na hodnotu stejnou nebo větší než počet segmentů přidělených každé instanci databázového serveru.

Další rady ke konfiguraci sdílené paměti operačního systému najdete v operačním systému UNIX v souboru machine notes (poznámky k počítači) nebo v souboru release notes (poznámky k verzi) v operačním systému Windows. Cestu k těmto souborům naleznete v úvodu této příručky.

Uvolnění sdílené paměti programem onmode -F

Databázový server automaticky neuvolňuje segmenty sdílené paměti přidané za běhu. Poté, co byla paměť jednou přidělena databázovému serveru, zůstává nedostupná pro všechny ostatní procesy spuštěné na hostitelském počítači. Když databázový server spustí velký dotaz pro podporu rozhodování, může dostat přiděleno velké množství sdílené paměti. Po zpracování dotazu už tuto sdílenou paměť databázový server nepotřebuje. Nicméně sdílená paměť, která byla serveru přidělena pro vyřízení dotazu, zůstane přiřazena i nadále virtuální části, přestože už není potřeba.

Příkaz **onmode -F** vyhledá a vrátí nepoužité osmikilobajtové bloky sdílemé paměti, které databázový server stále zabírá. Ačkoli je tento příkaz spuštěn pouze krátkou dobu (jednu nebo dvě sekundy), **onmode -F** dramaticky zpomaluje aktivitu uživatele. Typicky jsou k tomuto zpomalení méně náchylné systémy s více procesory nebo více virtuálními procesory.

Doporučujeme spouštět příkaz **onmode -F** ve chvílích nízkého zatížení počítače pomocí plánovací služby operačního systému (jako je například **cron** v operačním systému UNIX). Navíc zvažte spuštění tohoto obslužného programu po provedení úkolů, které podstatně zvyšují velikost sdílené paměti využívané databázovým serverem, jako jsou například velké dotazy pro podporu rozhodování, sestavování rejstříku, řazení nebo zálohování. Další informace o programu**onmode** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Konfigurační parametry, které ovlivňují využití paměti

Následující parametry konfigurace databázového serveru významně ovlivňují využití paměti:

- BUFFERPOOL
- DS_NONPDQ_QUERY_MEM
- DS_TOTAL_MEMORY

- EXTSHMADD
- LOCKS
- LOGBUFF
- MAX_RES_BUFFPCT
- PHYSBUFF
- RESIDENT
- SHMADD
- SHMBASE
- SHMTOTAL
- SHMVIRTSIZE
- SHMVIRT_ALLOCSEG
- STACKSIZE
- Parametry mezipaměti paměti (další informace naleznete v části “Parametry, které ovlivňují mezipaměti” na stránce 4-19)
- Velikost síťové vyrovnávací paměti (další informace naleznete v části “Společné oblasti vyrovnávacích pamětí v síti” na stránce 3-14)

Parametr SHMBASE označuje počáteční adresu sdílené paměti databázového serveru. Pokud je nastaven podle pokynů v souboru machine notes (poznámky k počítači) nebo release notes (poznámky k verzi) nemá tento parametr znatelný vliv na výkon. Cestu k těmto souborům naleznete v úvodu této příručky.

Parametr DS_NONPDQ_QUERY_MEM zvyšuje množství paměti dostupné pro jiné než PDQ dotazy Tento parametr můžete použít pouze v případě, že je nastavena priorita dotazů PDQ na nulu. Další informace naleznete v části “Více paměti pro dotazy se hashovacími spojeními, souhrny a dalšími prvky náročnými na paměť” na stránce 13-21.

Následující část popisuje vliv na výkon a úvahy spojené s některými konfiguračními parametry, vypsanými v úvodu této části. Další informace o těchto parametrech naleznete v části*IBM Informix Dynamic Server Administrator's Reference*.

Nastavení velikosti společné oblasti vyrovnávacích pamětí,vyrovnávací paměti logického protokolu a fyzického protokolu.

Hodnoty, které nastavíte pro konfigurační parametry BUFFERPOOL, DS_TOTAL_MEMORY, LOGBUFF a PHYSBUFF závisí na typu aplikace, kterou používáte (OLTP nebo DSS) a velikosti stránky. Tabulka 4-2 na stránce 4-8 vypíše doporučená nastavení těchto parametrů nebo pokyny k jejich nastavení.

Informace o zjišťování velikosti rezidentní části sdílené paměti najdete v části “Rezidentní část” na stránce 4-2. V tomto výpočtu figuruje velikost společné oblasti vyrovnávacích pamětí, vyrovnávací paměť logického protokolu, vyrovnávací paměť fyzického protokolu a tabulka zámků.

Tabulka 4-2. Pokyny k aplikacím OLTP a DSS

Parametr	aplikace OLTP	aplikace DSS
BUFFERPOOL	Procentuální část fyzické paměti, která je potřeba pro vyrovnávací paměť, závisí na množství paměti systému a množství paměti použité jinými aplikacemi.	Pro odlehčená prohledávání, dotazy a řazení nastavte malou velikost vyrovnávací paměti a zvyšte hodnotu parametru DS_TOTAL_MEMORY. Pro operace jako sestavování rejstříků, které čtou data přes společnou oblast vyrovnávacích pamětí, nastavte větší počet vyrovnávacích pamětí.
DS_TOTAL_MEMORY	Nastavte na 20 až 50 procent hodnoty SHMTOTAL v kB.	Nastavte na 50 až 90 procent hodnoty SHMTOTAL.
LOGBUFF	Pokud používáte protokolování bez vyrovnávací paměti nebo protokolování ANSI, použijte hodnotu **stránky/io** v sekci logický protokol výstupu **onstat -l** pro hodnotu LOGBUFF. Pokud používáte protokolování s vyrovnávací pamětí, nechte hodnotu **stránky/io** nízkou. Doporučená hodnota parametru LOGBUFF je mezi 16 a 32 kB nebo 64 kB pro vysoká pracovní zatížení.	Protože pro aplikace DSS bývá protokolování databáze nebo tabulky většinou vypnuté, nastavte hodnotu LOGBUFF na 32 kB.
PHYSBUFF	Pokud aplikace používá fyzické protokolování, zkontrolujte hodnotu **stránky/io** v sekci fyzický protokol výstupu **onstat -l** a ujistěte se, že vstupní a výstupní aktivita není příliš velká. Hodnotu parametru PHYSBUFF nastavte na hodnotu dělitelnou velikostí stránky. Doporučená hodnota parametru PHYSBUFF je 16 stránek.	Protože většina DSS aplikací nepoužívá fyzické protokolování, nastavte hodnotu parametru PHYSBUFF na 32 kB.

BUFFERPOOL

Pokud chcete větší délku klíče, než jaká je k dispozici pro výchozí velikost stránky, můžete zadat velikost stránky pro standardní nebo dočasný prostor dbspace. Kořenový prostor dbspace sestává ze stránek výchozí velikosti. Chcete-li určit velikost stránky, musí být velikost celým násobkem výchozí velikosti stránky a nemůže být větší než 16 kB. U systémů s dostatečnou velikostí paměti můžeme mezi výkonové výhody větší velikosti stránky počítat:

- Sníženou hloubku indexů b-stromu i pro menší klíče indexu.
- Sníženou dobu kontrolního bodu, k čemuž typicky dochází u větších velikostí stránky.

Chcete-li získat další výkonnostní výhody, postupujte takto:

- Seskupte stejně dlouhé řady, které by jinak zabíraly více stránek výchozí velikosti.

- Definujte jinou velikost stránky pro dočasné tabulky, takže dočasné tabulky budou mít samostatnou společnou oblast vyrovnávacích pamětí.

Konfigurační parametr BUFFERPOOL můžete použít k vytvoření společné oblasti vyrovnávacích pamětí, která odpovídá velikosti stránky prostoru dbspace. (Tímto způsobem můžete implementovat vlastní společnou oblast vyrovnávacích pamětí.) Můžete vytvořit společné oblasti vyrovnávacích pamětí s velikostmi stránek, které nejsou používány žádným prostorem dbspace.

Doporučení: Společnou oblast vyrovnávacích pamětí vytvořte dříve, než vytvoříte standardní nebo dočasný prostor dbspace s jinou než výchozí velikostí stránky.

Nelze vytvořit vícenásobnou společnou oblast vyrovnávacích pamětí se stejnou velikostí stránky.

Vytvoříte-li prostor dbspace s velikostí stránky, která neodpovídá společné oblasti vyrovnávacích pamětí, server Dynamic Server automaticky vytvoří společnou oblast vyrovnávacích pamětí pomocí výchozího parametru definovaného v konfiguračním souboru ONCONFIG.

Informace, které byly určovány pomocí konfiguračních parametrů BUFFERS, LRUS, LRU_MAX_DIRTY a LRU_MIN_DIRTY před verzí 10.0, jsou nyní určovány pomocí konfiguračního parametru BUFFERPOOL. Po vytvoření společné oblasti vyrovnávacích pamětí definujete informace o oblasti včetně její velikosti, počtu vyrovnávacích pamětí v oblasti, počet naposledy použitých položek (LRU) v oblasti a hodnoty parametrů **lru_min_dirty** a **lru_max_dirty**.

Změny automatického ladění LRU mají vliv na všechny společné oblasti vyrovnávacích pamětí a nastavují hodnoty **lru_min_dirty** a **lru_max_dirty** konfiguračního parametru BUFFERPOOL.

Neurčíte-li velikost stránky pro novou společnou oblast vyrovnávacích pamětí, použije server Dynamic Server výchozí velikost stránky operačního systému (4 kB v systému Windows a 2 kB na většině platforem se systémem UNIX) jako výchozí velikost stránky pro společnou oblast vyrovnávacích pamětí.

V následujícím příkladu jsou uvedeny informace o společné oblasti vyrovnávacích pamětí definované na operačním systému UNIX s výchozí velikostí stránky:

```
BUFFERPOOL default,lrus=8,buffers=5000,lru_min_dirty=50,lru_max_dirty=60
BUFFERPOOL size=2K,buffers=5000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```

Konfigurační soubor ONCONFIG může obsahovat více řádků BUFFERPOOL. Například v počítači s velikostí stránky 2kB může soubor ONCONFIG obsahovat až 9 řádek včetně výchozího nastavení.

Pole **buffers** konfiguračního parametru BUFFERPOOL určuje počet datových vyrovnávacích pamětí, které má databázový server k dispozici. Tyto vyrovnávací paměti jsou uloženy v rezidentní části sdílené paměti a jsou používány k rychlému dočasnému ukládání dat z databáze do paměti.

Pokud je hodnota **buffers** nastavena na nulu (0) nebo hodnota **buffers** v konfiguračním parametru BUFFERPOOL chybí úplně, dynamický server nevytvoří společnou oblast vyrovnávacích pamětí s určenou velikostí stránky.

Když je databázový server v režimu online, klidovém režimu nebo v režimu administrace, můžete použít obslužný program **onparams** a přidat novou společnou oblast vyrovnávacích

pamětí jiné velikosti. Pokud použijete obslužný program **onparams**, budou údaje, které jste určili pro společnou oblast vyrovnávacích pamětí, automaticky přeneseny do so uboru ONCONFIG.

Pokud použijete obslužný program **onparams**, zadejte údaje následovně:

```
onparams -b -g <velikost stránky vyrovnávací paměti v KB> -n
<počet vyrovnávacích pamětí>
-r <počet front LRU> -x <parametr max dirty
 (jsou povoleny neceločíselné hodnoty)>
-m <parametr minimum dirty (jsou povoleny neceločíselné hodnoty)>
```

Například:

```
onparams -b -g 6 -n 3000 -r 2 -x 2.0 -m 1.0
```

Podle tohoto řádku bude vytvořeno 3000 vyrovnávacích pamětí, každá o velikosti 6 kB se dvěma frontami LRU, hodnotou **lru_max_dirty** nastavenou na 2 procenta a hodnotou **lru_min_dirty** nastavenou na 1 procento.

Konfigurační parametr BUFFERPOOL má významný vliv na vstupně-výstupní operace databáze a propustnost transakcí. Čím více vyrovnávacích pamětí je k dispozici, tím je pravděpodobnější, že potřebná stránka s daty už může být uložena v paměti jako výsledek předchozího požadavku. Každopádně přidělení příliš mnoha vyrovnávacích pamětí může mít vliv na systém správy paměti a vést k nadměrnému stránkování operačního systému.

Databázové servery používají následující vzorec k výpočtu množství paměti pro přidělení této datové společné oblasti vyrovnávacích pamětí:

```
bufferpoolsize = BUFFERS * page_size
```

page_size je v operačním systému velikost stránky v paměti, obslužný program onstat spuštěný s volbou b (**onstat -b**) ji zobrazuje na posledním řádku v poli **velikost vyrovnávací paměti**.

```
onstat -b
...

Buffers
address   userthread flgs pagenum  memaddr  nslots  pgflgs xflgs owner  waitlist
2 modified, 0 resident, 200 total, 256 hash buckets, 4096 buffer size
```

V systému Windows je velikost stránky vždy 4 kB (4096 B).

Procentuální část fyzické paměti, která je potřeba pro vyrovnávací paměť, závisí na množství paměti systému a množství paměti použité jinými aplikacemi. U systémů s velkou fyzickou pamětí (4 GB nebo více) může zabírat prostor pro vyrovnávací paměti až 90 procent fyzické paměti. U systému s menší fyzickou pamětí může prostor pro vyrovnávací paměti zabírat 20 až 25 procent fyzické paměti.

Všechny ostatní parametry sdílené paměti vypočítejte po nastavení prostoru pro vyrovnávací paměť (**buffers** *bufferpool_page_size*). Hodnota *bufferpool_page_size* je velikost pole konfiguračního parametru BUFFERPOOL.

Pokud má například systém velikost stránky 2 kB a 100 MB fyzické paměti, můžete nastavit hodnotu v poli **buffers** v rozmezí 10 000 až 12 500, což bude mít za následek přidělení 20 až 25 MB fyzické paměti.

Poznámka: Používá-li systém jiné než výchozí velikosti stránky, bude pravděpodobně potřeba zvýšit velikost fyzického protokolu. Pokud systém provádí mnoho aktualizací stránek jiné velikosti než výchozí, je možné, že bude potřeba zvýšit velikost fyzického protokolu o 150 až 200 %. Vyladění velikosti fyzického

protokolu může vyžadovat několik pokusů. Velikost fyzického protokolu můžete nastavit podle potřeby v závislosti na tom, jak často zaplnění fyzického protokolu spouští kontrolní body.

Další informace o vytváření prostoru dbspace s jinou než výchozí velikostí stránky a další informace o definici společné oblasti vyrovnávacích pamětí, najdete v části *Příručka administrátora serveru IBM Informix Dynamic Server*. Další informace o konfiguračním parametru BUFFERPOOL a informace jak používat obslužný program **onparams** najdete v části *IBM Informix Dynamic Server Administrator's Reference*.

64bitové adresování a vyrovnávací paměti: Chcete-li využít výhod velké paměti v počítačích se 64bitovým adresováním, zvyšte počet vyrovnávacích pamětí ve společné oblasti vyrovnávacích pamětí. Zvětšením společné oblasti vyrovnávacích pamětí se zvyšuje pravděpodobnost, že potřebná stránka bude už v paměti uložena.

Inteligentní velké objekty a vyrovnávací paměť: Při výchozím nastavení databázový server načítá inteligentní velké objekty do vyrovnávacích pamětí v rezidentní části sdílené paměti (známé také jako společná oblast vyrovnávací paměti).

V závislosti na situaci můžete zvýšit výkon aplikací používajících inteligentní velké objekty jednou z následujících akcí:

- Při výchozím nastavení použijte společnou oblast vyrovnávací paměti a zvyšte hodnotu proměnné **buffers** konfiguračního parametru BUFFERPOOL.

  Pokud aplikace často přistupují k inteligentním velkým objektům o velikosti 2 nebo 4 kilobajty, použijte společnou oblast vyrovnávacích pamětí, čímž zůstanou v paměti déle.

  Podle následujícího vzorce zvyšte hodnotu buffers:

  ```
  Additional_BUFFERS = numcur_open_lo *
                       (lo_userdata / pagesize)
  ```

  numcur_open_lo
  : je počet souběžně otevřených inteligentních velkých objektů, které můžete získat pomocí volby **onstat -g smb fdd**.

  lo_userdata
  : je počet bajtů dat inteligentního velkého objektu, které chcete uložit do mezipaměti.

  pagesize
  : je velikost stránky v bajtech pro databázový server.

  Obecné pravidlo určuje, že by měl být dostupný dostatek vyrovnávacích pamětí k uložení dvou stránek každého otevřeného inteligentního velkého objektu. (Další stránka je dostupná pro pro účely dopředného čtení).

- Odlehčené vstupně-výstupní vyrovnávací paměti použijte ve virtuální části sdílené paměti.

  Odlehčené vstupně-výstupní vyrovnávací paměti použijte pouze v případě čtení nebo zápisu inteligentních velkých objektů při operacích větších než 8000 bajtů a pokud k nim přistupujete zřídka. To znamená odlehčené, vstupně-výstupní vyrovnávací paměti použijte pokud funkce pro zápis nebo čtení volá čtení velkého množství dat a to při volání pouze jedné funkce.

  Při použití odlehčené vstupně-výstupní vyrovnávací paměti zabráníte možnému přeplnění společné oblasti vyrovnávací paměti inteligentními velkými objekty a budete mít k dispozici více vyrovnávacích pamětí pro ostatní stránky dat, ke kterým často přistupuje více uživatelů. Další informace naleznete v části “Odlehčený vstup - výstup pro inteligentní velké objekty” na stránce 5-20.

Monitorování vyrovnávacích pamětí: Aktivitu vyrovnávacích pamětí a společných oblastí vyrovnávacích pamětí můžete monitorovat pomocí následujících voleb obslužného programu **onstat**:

- **Pomocí voleb -b** a **-B** lze zobrazit obecné informace o vyrovnávací paměti

- **Pomocí volby -R** lze zobrazit statistiky fronty LRU.
- **Pomocí volby -X** lze zobrazit informace o vstupně-výstupních jednotkových procesech, čekajících na vyrovnávací paměť.
- Pomocí volby **-p** lze zobrazit informace o stránce včetně statistických údajů společné oblasti vyrovnávací paměti.

Obslužný program onstat s volbou -p (**onstat -p**) lze použít k monitorování rychlosti čtení z mezipaměti společné oblasti vyrovnávací paměti. Tato rychlost znamená procentní část stránek databáze, které jsou při požadavku na stránku uloženy ve vyrovnávací paměti sdílené paměti. (Pokud stránka ještě není uložena databázový server ji musí do paměti zkopírovat z disku.) Pokud databázový server stránku najde ve společné oblasti vyrovnávací paměti, stráví méně času vstupně-výstupními operacemi disku. Z toho důvodu byste pro vysoký výkon měli chtít vysokou rychlost čtení z mezipaměti. U aplikací OTLP, kde spousta uživatelů požaduje čtení malých množství dat, je cílem dosáhnout rychlosti čtení z mezipaměti až 95 procent nebo více.

Pokud je rychlost čtení z mezipaměti paměti nízká, můžete opakovaně zvýšit počet vyrovnávacích pamětí a restartovat databázový server. Když zvýšíte hodnotu BUFFERPOOL proměnné buffers, postupně dosáhnete hodnoty, při které další zvyšování hodnoty nepřináší další znatelný vzestup rychlosti čtení z mezipaměti nebo dosáhnete horního limitu přidělení sdílené paměti operačního systému.

Ke zjištění úrovně prohledávání stránek a aktivity stránkování použijte podpůrný program operačního systému pro monitorování správy paměti (jako je například **vmstat** nebo **sar** v systému UNIX) to note the level of page scans and paging-out activity. Pokud tyto úrovně najednou vzrostou, nebo vzrostou během vysoké aktivity databáze nad přijatelné hodnoty snižte hodnotu BUFFERPOOL proměnné buffers.

DS_TOTAL_MEMORY

Parametr DS_TOTAL_MEMORY vytvoří strop pro množství sdílené paměti, kterou může získat jeden dotaz. Tento parametr můžete použít k omezení vlivu velkých a paměťově náročných dotazů na výkon. Čím větší nastavíte hodnotu tohoto parametru, tím více paměti bude moci velký dotaz použít a tím méně paměti zůstane k dispozici pro zpracovávání ostatních dotazů a transakcí.

Pro OLTP aplikace nastavte parametr DS_TOTAL_MEMORY na hodnotu mezi 20 a 50 procenty hodnoty SHMTOTAL, v kilobajtech. Pro aplikace, které zahrnují velké dotazy pro podporu rozhodování, zvyšte parametr DS_TOTAL_MEMORY na hodnotu mezi 50 a 80 procenty hodnoty SHMTOTAL. Pokud používáte databázový server exkluzivně pro dotazy pro podporu rozhodování, nastavte tento parametr na 90 procent hodnoty SHMTOTAL.

Jednotka kvanta je minimální přírůstek paměti přidělené dotazu. Správce přidělující paměť (Memory Grant Manager (MGM)) přiděluje paměť dotazům po jednotkách kvanta. Databázový server používá hodnotu parametru DS_MAX_QUERIES a parametru DS_TOTAL_MEMORY k výpočtu kvanta paměti podle následujícího vzorce:

```
quantum = DS_TOTAL_MEMORY / DS_MAX_QUERIES
```

Chcete-li povolit více současných dotazů s menším kvantem paměti, doporučujeme zvýšit hodnotu parametru DS_MAX_QUERIES. Další informace o parametru DS_MAX_QUERIES najdete v části “Omezení dopadu dotazů s intenzivním využitím CPU na výkon pomocí konfiguračního parametru DS_MAX_QUERIES” na stránce 3-11. Další informace o MGM najdete v části “Správce přidělující paměť” na stránce 12-6.

Algoritmus pro určení hodnoty DS_TOTAL_MEMORY: Databázový server si hodnotu parametru DS_TOTAL_MEMORY odvodí, pokud ji nenastavíte, nebo pokud nastavíte

nevhodnou hodnotu. Kdykoli databázový server změní hodnotu, kterou jste přiřadili parametru DS_TOTAL_MEMORY, je do konzole zapsána následující zpráva:

```
Hodnota DS_TOTAL_MEMORY přepočítána a změněna z původní hodnoty původní hodnota kB
            na nová hodnota kB
```

Proměnná *původní hodnota* reprezentuje hodnotu, kterou jste přiřadili parametru DS_TOTAL_MEMORY v konfiguračním souboru Proměnná *nová hodnota* reprezentuje hodnotu, kterou databázový server odvodil.

Pokud se zobrazí předchozí zpráva, můžete podle algoritmu zjistit, které hodnoty považuje databázový server za nevhodné. Následně můžete hodnoty na základě tohoto zjištění opravit.

Následující část dokumentuje algoritmus, který databázový server používá k odvození nové hodnoty parametru DS_TOTAL_MEMORY.

Odvození minimální hodnoty paměti pro podporu rozhodování: V první části algoritmu databázový server vytvoří minimální hodnotu pro paměť pro podporu rozhodování. Když nastavíte hodnotu konfiguračního parametru DS_MAX_QUERIES, databázový server určí minimální množství paměti pro podporu rozhodování podle následujícícho vzorce:

```
min_ds_total_memory = DS_MAX_QUERIES * 128 kB
```

Pokud nenastavíte hodnotu parametru DS_MAX_QUERIES, databázový server použije následující vzorec založený na hodnotě parametru VPCLASS cpu nebo NUMCPUVPS:

```
min_ds_total_memory = NUMCPUVPS * 2 * 128 kB
```

Odvození funkční hodnoty paměti pro podporu rozhodování: V druhé části algoritmu databázový server vytvoří funkční hodnotu pro množství paměti pro podporu rozhodování. Databázový server tuto hodnotu ověří ve třetí a poslední část algoritmu.

Pokud je nastavena hodnota DS_TOTAL_MEMORY: Databázový server nejprve zkontroluje, zda je nastavena hodnota parametru SHMTOTAL. Pokud je hodnota SHMTOTAL nastavena, databázový server použije k výpočtu množství paměti pro podporu rozhodování následující vzorec:

```
IF DS_TOTAL_MEMORY <= SHMTOTAL - nondecision_support_memory THEN
   decision_support_memory = DS_TOTAL_MEMORY
ELSE
    decision_support_memory = SHMTOTAL -
                  nondecision_support_memory
```

Tento algoritmus efektivně zabraňuje nastavení parametru DS_TOTAL_MEMORY na hodnoty, které databázový server nemůže efektivně přidělit paměti pro podporu rozhodování.

Pokud není hodnota parametru SHMTOTAL nastavena, databázový server nastaví množství paměti pro podporu rozhodování na stejnou hodnotu, kterou jste zadali v parametru DS_TOTAL_MEMORY.

Pokud není nastavena hodnota parametru DS_TOTAL_MEMORY: Pokud nenastavíte hodnotu parametru DS_TOTAL_MEMORY, bude databázový server postupovat následovně. Nejdříve zkontroluje, zda je nastavena hodnota parametru SHMTOTAL. Pokud je hodnota SHMTOTAL nastavena, databázový server použije k výpočtu množství paměti pro podporu rozhodování následující vzorec:

```
decision_support_memory = SHMTOTAL -
                  nondecision_support_memory
```

Pokud nebyla nastavena hodnota parametru SHMTOTAL, databázový server nastaví nastaví paměť pro podporu rozhodování jako v následujícím příkladu:

```
decision_support_memory = min_ds_total_memory
```

Další informace o proměnné min_ds_total_memory najdete v části “Odvození minimální hodnoty paměti pro podporu rozhodování” na stránce 4-13.

Kontrola odvozené paměti pro podporu rozhodování: Závěrečná část algoritmu zkontroluje, že je množství sdílené paměti větší než hodnota *min_ds_total_memory* a menší než velikost celkového paměťového prostoru počítače. Pokud je odvozená velikost paměti pro podporu rozhodování menší než hodnota proměnné *min_ds_total_memory* , bude velikost paměti pro podporu rozhodování nastavena na stejnou hodnotu jako proměnná *min_ds_total_memory*.

Pokud je odvozená velikost paměti pro podporu rozhodování větší než celkový paměťový prostor počítače, bude velikost paměti pro podporu rozhodování nastavena na maximální velikost paměti počítače.

LOGBUFF

Parametr LOGBUFF určuje množství sdílené paměti rezervované pro všechny tři vyrovnávací paměti, které udržují záznamy logického protokolu před vyprázdněním do souboru logického protokolu na disk. Velikost vyrovnávacích pamětí určuje, jak často budou naplněny a jak často tedy musí být vyprázdněny do souboru logického protokolu na disku.

V případě protokolování inteligentních velkých objektů zvětšete velikost vyrovnávacích pamětí logických protokolů, čímž zabráníte častému vyprazdňování vyrovnávacích pamětí do souboru logického protokolu na disku.

PHYSBUFF

Parametr PHYSBUFF určuje množství sdílené paměti rezervované pro obě vyrovnávací paměti, které slouží jako dočasný úložný prostor pro stránky dat, které mají být změněny. Velikost vyrovnávacích pamětí určuje, jak často budou naplněny a jak často tedy musí být vyprázdněny do souboru fyzického protokolu na disku. Pro parametr PHYSBUFF zvolte hodnotu, která je sudým násobkem velikosti stránky systému.

LOCKS

Počáteční velikost tabulky zámků určuje konfigurační parametr LOCKS. Tabulka zámků udržuje záznam pro každý zámek, který relace používá. Pokud počet zámků přidělených relaci překročí hodnotu parametru LOCKS, databázový server zdvojnásobí velikost tabulky zámků. Po každém zdvojnásobení velikosti tabulky zámků není přiděleno více než 100 000 zámků. Databázový server může tabulku dynamicky zvětšit až patnáctkrát.

Každý zámek v tabulce zámků zabírá 120 bajtů. S tímto potřebným množstvím paměti musíte počítat při konfiguraci sdílené paměti.

Maximální hodnota parametru LOCKS je:

- 500,000,000 na 64-bitovém systému.
- 8,000,000 na 32-bitovém systému. Celkový maximální počet zámků databázového serveru je 9.500.000, což je 8.000.000 plus 15 dynamických přidělení po 100.000 zámcích.

Výchozí hodnota konfiguračního parametru LOCKS je 2000. Další informace o změně této výchozí hodnoty najdete v části “Konfigurace a sledování počtu zámků” na stránce 8-11.

Chcete-li odhadnout jinou hodnotu konfiguračního parametru LOCKS, odhadněte maximální počet zámků, které bude dotaz potřebovat, a vynásobte tuto hodnotu počtem souběžně pracujících uživatelů. K odhadu počtu zámků, které dotaz potřebuje můžete použít rady v následující tabulce.

Počet zámků na pří kaz	Úroveň izolace	Tabulka	Řádek	Klíč	Textová nebo binární data	Data typu CLOB a BLOB
SELECT	Neaktualizované čtení	0	0	0	0	0
SELECT	Potvrzené čtení	1	0	0	0	0
SELECT	Kurzorová stabilita	1	1	0	0	1 zámek pro hodnotu CLOB nebo BLOB nebo (pokud jsou použity zámky rozsahu byte) 1 zámek pro každý rozsah
SELECT	Indexované opakované čtení	1	Počet řádků odpovídajících podmínkám	Počet řádků odpovídajících podmínkám	0	1 zámek pro hodnotu CLOB nebo BLOB nebo (pokud jsou použity zámky rozsahu byte) 1 zámek pro každý rozsah
SELECT	Postupné opakované čtení	1	0	0	0	1 zámek pro hodnotu CLOB nebo BLOB nebo (pokud jsou použity zámky rozsahu byte) 1 zámek pro každý rozsah
INSERT	Nelze použít	1	1	Počet indexů	Počet stránek textových nebo binárních dat	1 zámek pro hodnotu CLOB nebo BLOB
DELETE	Nelze použít	1	1	Počet indexů	Počet stránek textových nebo binárních dat	1 zámek pro hodnotu CLOB nebo BLOB
UPDATE	Nelze použít	1	1	2 na každou hodnotu změněného klíče	Počet stránek textových nebo binárních dat plus počet stránek původních dat	1 zámek pro hodnotu CLOB nebo BLOB nebo (pokud jsou použity zámky rozsahu byte) 1 zámek pro každý rozsah

Důležité: Během provádění příkazu SQL DROP DATABASE databázový server získává a udržuje zámek pro každou tabulku v databázi, dokud operace DROP neskončí. Zkontrolujte, zda je hodnota parametru LOCKS dostatečně velká na to, aby bylo možno uchovat největší počet tabulek v databázi.

RESIDENT

Parametr RESIDENT určuje, zda je vynucena rezidence sdílené paměti v rezidentní části sdílené paměti databázového serveru. Tento parametr je funkční pouze v počítačích, které podporují vynucenou rezidenci. Rezidentní část databázového serveru obsahuje společné oblasti vyrovnávacích pamětí využívané pro čtení a zápis do databáze. Pokud tyto vyrovnávací paměti zůstanou ve fyzické paměti, výkon se zvýší. Doporučujeme nastavit parametr RESIDENT na hodnotu 1. Pokud na vašem počítači volby vynucené rezidence není, pak databázový server měl s tímto parametrem pravděpodobně problémy a nadále jej ignoruje.

V počítačích s podporou 64bitového adresování může být společná oblast vyrovnávací paměti velká, stejně jako virtuální část sdílené paměti databázového serveru. Virtuální část obsahuje různé části mezipamětí, které zvyšují výkon při zpracovávání více dotazů zároveň, které přistupují ke stejné tabulce. (Další informace naleznete v části “Parametry, které

ovlivňují mezipaměti" na stránce 4-19). Chcete-li, aby virtuální část paměti zůstala rezidentní v paměti fyzické, nastavte parametr RESIDENT na hodnotu -1.

Pokud je společná oblast vyrovnávací paměti příliš velká, ale fyzická paměť tak velké není, můžete nastavit parametr RESIDENT na hodnotu větší než 1, čímž určíte počet paměťových segmentů, které mají zůstat ve fyzické paměti. Po tomto určení bude v paměti rezidentní pouze část společné oblasti vyrovnávací paměti.

Rezidenci rezidentní části sdílené paměti můžete zapnout nebo vypnout následujícími způsoby:

- Pomocí obslužného programu **onmode** můžete dočasně obrátit rezidenci sdílené paměti za běhu databázového serveru.
- Změnou hodnoty parametru RESIDENT můžete rezidenci sdílené paměti vypnout nebo zapnout. Změna se projeví po novém spuštění sdílené paměti databázového serveru.

SHMADD a EXTSHMADD

Konfigurační parametr SHMADD určuje velikost každého přírůstku sdílené paměti, který databázový server dynamicky přidá k virtuální části. Konfigurační parametr EXTSHMADD určuje velikost přidaného rozšiřujícího segmentu.

Určení velikosti přírůstku je dílem kompromisu. Přidávání sdílené paměti spotřebovává čas procesoru. Čím větší je každý přírůstek, tím méně často je potřeba zatěžovat přidáváním procesor, nicméně je potom k dispozici méně paměti pro ostatní procesy. Obecně se dává přednost přidávání velkých přírůstků, ale při velkém zatížení paměti (vysoká rychlost prohledávání nebo stránkování) umožňují menší přírůstky efektivnější sdílení paměťových prostředků mezi souběžně spuštěnými programy.

Rozsah hodnot parametru SHMADD je od 1024 do 4294967296 kB na 64bitovém databázovém serveru a od 1024 do 524288 na 32bitovém databázovém serveru. Následující tabulka obsahuje doporučení pro nastavení parametru SHMADD podle velikosti fyzické paměti.

Velikost paměti	Hodnota SHMADD
256 MB nebo méně	8192 kB (výchozí hodnota)
257 až 512 MB	16,384 kB
více než 512 MB	32768 kB

Rozsah hodnot parametru EXTSHMADD je od 1024 do 524288.

Poznámka: Segment sdílené paměti může být velký až 4 TB, v závislosti na omezení platformy a hodnotě SHMMAX parametru jádra. Pomocí příkazu **onstat -g seg** zobrazíte počet segmentů sdílené paměti, které databázový server právě používá.

Další informace o konfiguraci segmentů sdílené paměti najdete v části "Konfigurace sdílené paměti UNIX" na stránce 4-5. Další informace o parametrech SHMADD, SHMMAX a EXTSHMADD najdete v části *IBM Informix Dynamic Server Administrator's Reference*.

SHMTOTAL

Parametr SHMTOTAL určuje absolutní limit množství sdílené paměti, které může jedna instance databázového serveru použít. Pokud je parametr SHMTOTAL nastaven na hodnotu 0 nebo není nastaven vůbec, databázový server pokračuje v připojování další sdílené paměti podle potřeby, až do vyčerpání veškeré virtuální paměti systému.

Parametr SHMTOTAL můžete obvykle nechat nastavený na hodnotu 0 mimo následujících případů:

- Je nutné omezit množství paměti využívané databázovým serverem kvůli souběžně spuštěným aplikacím nebo z jiných důvodů.
- Operačnímu systému došel odkládací prostor a nechová se normálně.

Ve druhém případě můžete nastavit parametr SHMTOTAL na hodnotu, která je o několik MB nižší než celkový odkládací prostor počítače.

SHMVIRTSIZE

Parametr SHMVIRTSIZE určuje velikost virtuální části sdílené paměti, kterou je možno přidělit po spuštění databázového serveru. Virtuální část sdílené paměti uchovává jak informace specifické pro relaci a požadavek, tak ostatní informace.

Ačkoli databázový server zvětšuje virtuální část sdílené paměti podle požadavků na zpracování velkých dotazů nebo zátěžových špiček, přidělování sdílené paměti prodlužuje dobu zpracování požadavku. Proto doporučujeme nastavit parametr SHMVIRTSIZE na hodnotu, která postačí pro pokrytí běžných denních požadavků. Hodnota parametru SHMVIRRTSIZE může nabývat až hodnoty parametru SHMMAX.

Maximální hodnota kladné celé hodnoty parametru SHMVIRTSIZE je:

- 4 TB na 64bitovém databázovém serveru
- 2 GB na 32bitovém databázovém serveru

Jako počáteční nastavení doporučujeme použít větší z následujících hodnot:

- 8000
- *connections* * 350

Proměnná *connections* určuje počet spojení pro všechny typy sítí, které jsou určeny konfiguračním parametrem NETTYPE v souboru **sqlhosts** nebo v registrech. (Databázový server používá jako výchozí hodnotu *connections* * 200.)

Poté, co využití systému dosáhne stabilní hodnoty zatížení, můžete znovu konfigurovat hodnotu parametru SHMVIRTSIZE. Jak je zmíněno v části “Uvolnění sdílené paměti programem onmode -F” na stránce 4-6, můžete nastavit databázový server tak, aby segmenty sdílené paměti, které se již se po zpracování velkých dotazů nebo zátěžových špiček nepoužívají, uvolňoval.

SHMVIRT_ALLOCSEG

Konfigurační parametr SHMVIRT_ALLOCSEG určuje:

- Prahovou hodnotu, při které by měla být databázovému serveru přidělena další paměť a
- Výstražnou událost bezpečnostního kódu, která je aktivována v případě, že serveru nemůže být přidělen další paměťový segment.

SHMVIRT_ALLOCSEG zajišťuje, že server nikdy nebude mít nedostatek paměti.

Když nastavíte tento konfigurační parametr, musíte:

- Určit procentuální část použité paměti, nebo počet kilobajtů (celé číslo) zbývající paměti. Nelze použít záporné hodnoty a hodnoty mezi 0 a 0,39.
- Určit výstražnou událost bezpečnostního kódu, což je hodnota mezi 1 (nestojí za povšimnutí) a 5 (kritické). Pokud neurčíte událost bezpečnostního kódu, server nastaví hodnotu na 3, což je hodnota výchozí.

Příklad 1:

```
SHMVIRT_ALLOCSEG 3000, 4
```

Tento parametr určuje, že pokud bude ve virtuální paměti zbývat pro databázový server pouze 3000 kB volného místa a další část paměti nelze přidělit, server spustí výstražný kód úrovně 4.

Příklad 2:

```
SHMVIRT_ALLOCSEG .8, 4
```

Tento parametr určuje, že pokud bude pro databázovýserver zbývat pouze 20 procent paměti a další paměť nemůže být přidělena, server spustí výstražný kód úrovně 4.

Informace bezpečnostních kódech události najdete v části *IBM Informix Dynamic Server Administrator's Reference*.

STACKSIZE

Parametr STACKSIZE určuje výchozí velikost zásobníku pro každý jednotkový proces. Databázový server přiřadí každému aktivnímu jednotkovému procesu prostor určený tímto parametrem. Tento prostor se nachází ve virtuální části sdílené paměti databázového serveru.

Chcete-li snížit množství sdílené paměti, kterou databázový server dynamicky přidává, odhadněte množství prostoru potřebného pro zásobník pro průměrný počet jednotkových procesů, které jsou v systému spuštěny a toto množství použijte v hodnotě, kterou nastavíte parametru SHMVIRTSIZE. K odhadu prostoru potřebného pro zásobník použijte následující vzorec:

```
stacktotal = STACKSIZE * avg_no_of_threads
```

avg_no_of_threads vyjadřuje průměrný počet jednotkových procesů. Počet aktivních jednotkových procesů můžete monitorovat v pravidelných časových intervalech a zjistit tak průměrnou hodnotu. Ke sledování využití zásobníku jednotlivými procesy použijte příkaz **onstat -g sts** . Obecný odhad se pohybuje mezi 60 a 70 procenty celkového počtu spojení (určeno v parametru NETTYPE v souboru ONCONFIG) v závislosti na zatížení.

Databázový server provádí také uživatelské rutiny (UDR) pomocí uživatelských jednotkových procesů, které tento zásobník používají. Programátoři, kteří píší uživatelské rutiny by měli brát ohled na následující ukazatele, aby nedošlo k přeplnění zásobníku:

- Nepoužívat velká automatická pole.
- Vyvarovat se příliš hlubokému rekurzivnímu volání.

DB-Access

- Spravovat rekurzivní volání můžete pomocí **mi_call**.

Konec DB-Access

Pokud těmito způsoby nemůžete zabránit přetečení zásobníku, použijte modifikátor STACK příkazu CREATE FUNCTION, kterým můžete zvětšit velikost zásobníku pro určitou rutinu. Další informace o příkazu CREATE FUNCTION naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Parametry, které ovlivňují mezipaměti

Databázový server používá mezipaměť k ukládání informací do paměti místo provádění operací čtení z disku nebo zápisu na disk. Tyto mezipaměti zvyšují výkon pro více současných dotazů, které přistupují ke stejným tabulkám

Můžete určit konfigurační parametry a vyladit efektivitu každé mezipaměti,jak ukazuje následující tabulka.

Název mezipaměti	Popis mezipaměti	Konfigurační parametr a Popis
Data Dictionary (Datový slovník)	Uchovává informace o definici tabulky (jako například názvy sloupců a typy dat). Další informace najdete v části "Mezipaměť datového slovníku" na stránce 4-19.	DD_HASHSIZE: Počet sektorů v mezipaměti datového slovníku. DD_HASHMAX: Počet tabulek, které může databázový server uložit do jednoho sektoru.
Data Distribution (Rozdělení dat)	Ukládá statistiky rozdělení sloupce. Další informace najdete v části "Mezipaměť distribuce dat" na stránce 4-21.	DS_POOLSIZE: Celkový počet statistik rozdělení sloupce, které může databázový server uložit do jedné mezipaměti rozdělení dat. DPS_HASHSIZE: Počet sektorů v mezipaměti rozdělení dat.
SQL Statement (Paměť příkazů SQL)	Uchovává analyzované a optimalizované příkazy SQL Další informace najdete v části "Konfigurace mezipaměti příkazů SQL" na stránce 4-25.	STMT_CACHE: Povolení mezipaměti příkazů SQL. STMT_CACHE_HITS: Počet zpracování příkazu SQL, než je uložen do mezipaměti. STMT_CACHE_NOLIMIT: Zakáže ukládání do mezipaměti příkazů SQL v případě, že velikost přidělené paměti překročí hodnotu parametru STMT_CACHE_SIZE STMT_CACHE_NUMPOOL: Počet společných oblastí paměti pro mezipaměť příkazů SQL STMT_CACHE_SIZE: Velikost mezipaměti příkazů SQL v kilobajtech
Mezipaměť uživatelských rutin	Ukládá často používané rutiny UDR (rutiny SPL a externí rutiny). Další informace najdete v části "Mezipaměť uživatelských rutin" na stránce 10-31.	Celkový počet uživatelských rutin a rutin SPL v mezipaměti paměti uživatelských rutin Počet sektorů v mezipaměti uživatelských rutin

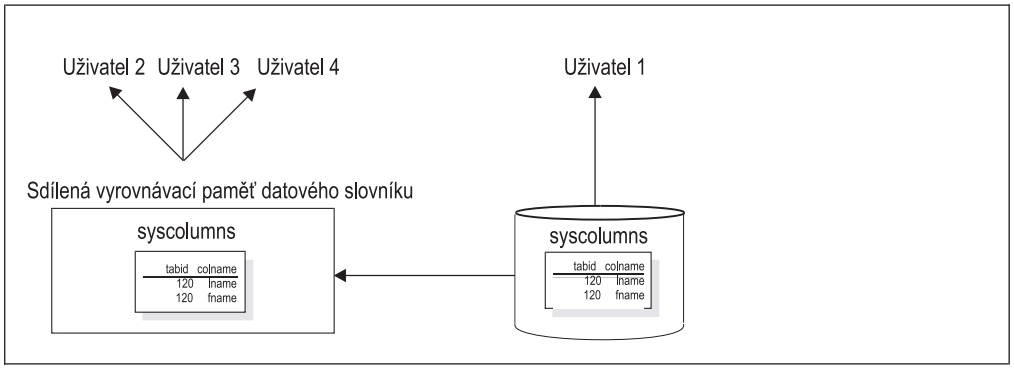
Mezipaměť pro uživatelské rutiny

Mezipaměť uživatelských rutin ukládá často používané uživatelské rutiny (rutiny SPL a externí rutiny). Konfigurační parametr PC_POOLSIZE určuje velikost mezipaměti uživatelských rutin. Další informace o mezipaměti uživatelských rutin a parametru PC_POOLSIZE najdete v části "Mezipaměť uživatelských rutin" na stránce 10-31.

Mezipaměť datového slovníku

Po prvním přístupu k tabulce získá databázový server informace o tabulce, které potřebuje (jako jsou například názvy sloupců a typy dat), ze systémové tabulky katalogu na disku. Po prvním přístupu k tabulce si databázový server tyto informace uloží do mezipaměti datového slovníku ve sdílené paměti.

Obrázek 4-1 zobrazuje, jak databázový server používá tuto mezipaměť pro více uživatelů současně. Uživatel 1 přistupuje poprvé k informacím o sloupcích tabulky **tabid 120**. Databázový server uloží informace o sloupci do mezipaměti datového slovníku. Když uživatelé uživatel 2, uživatel 3 a uživatel 4 později přistupují ke stejné tabulce,databázový server nemusí pro získání informací o tabulce číst data z disku. Místo toho přečte data o tabulce z mezipaměti datového slovníku ve fyzické paměti.



Obrázek 4-1. Mezipaměť datového slovníku

Databázový server stále ukládá stránky tabulky katalogu systému do společné oblasti vyrovnávací paměti, jako to dělá se všemi datovými a indexovými stránkami. Ale mezipaměť datového slovníku poskytuje další výkonnostní výhodu, protože informace datového slovníku jsou tříděny v mnohem efektivnějším formátu a organizovány pro rychlé vyhledávání.

Konfigurace datového slovníku

Databázový server používá pro ukládání a umisťování informací v mezipaměti datového slovníku hashovací algoritmus. Konfigurační parametry DD_HASHSIZE a DD_HASHMAX řídí velikost mezipaměti datového slovníku. Chcete-li změnit počet sektorů mezipaměti datového slovníku změňte hodnotu parametru DD_HASHSIZE (hodnota musí být prvočíslo). Chcete-li změnit počet tabulek, které lze uložit do jednoho sektoru, změňte hodnotu parametru DD_HASHMAX.

U středních až velkých systémů můžete začít s následujícími hodnotami těchto konfiguračních parametrů:

- DD_HASHSIZE 503
- DD_HASHMAX 4

S těmito hodnotami můžete potenciálně do mezipaměti datového slovníku uložit informace o 2012 tabulkách a v každém sektoru hashovací tabulky mohou být uloženy informace o maximálně 4 tabulkách.

Pokud sektor dosáhne maximální velikosti, databázový server použije mechanizmus nejdéle nepoužité položky a některé záznamy z datového slovníku vymaže.

Monitorování mezipaměti datového slovníku

K monitorování mezipaměti datového slovníku použijte příkaz **onstat -g dic**. Pokud nejsou v mezipaměti datového slovníku vypsány často používané tabulky, zkuste zvětšit její velikost.

Obrázek 4-2 zobrazuje vzorový výstup příkazu **onstat -g dic**.

```
Dictionary Cache: Number of lists: 31, Maximum list size: 10

list#  size  refcnt  dirty?  heapptr        table name
--------------------------------------------------------
   9     1      0      no     a210330   dawn@atlanta:informix.sysprocedures

  16     1      0      no     a46a420   dawn@atlanta:informix.orders

Total number of dictionary entries: 2
```

Obrázek 4-2. Výstup příkazu onstat -g dic

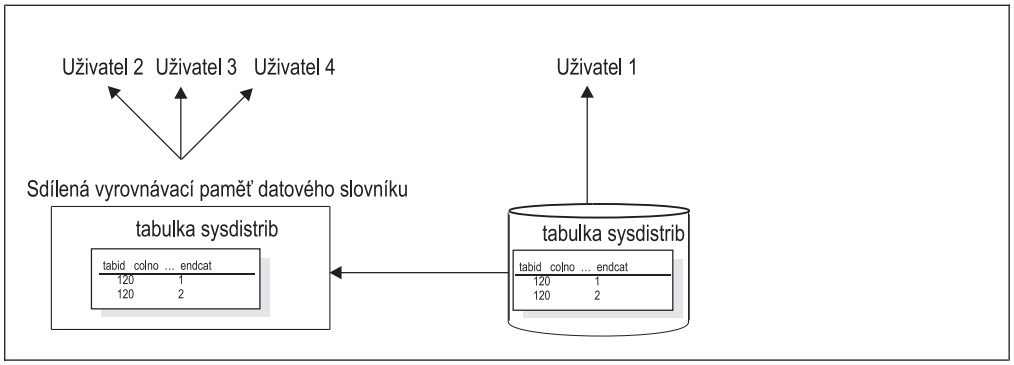
Výstup příkazu **onstat -g dic** má následující pole.

Pole	Popis
Number of Lists	Počet sektorů určený parametrem DD_HASHSIZE.
Maximum List Size	Počet tabulek povolených v každém sektoru.
List #	Počet sektorů.
Size	Počet tabulek v sektoru.
Ref cnt	Počet, kolikrát uživatel využil informace datového slovníku z mezipaměti pro tuto tabulku.
Dirty	Označení, zda už jsou informace datového slovníku neplatná.
Heap ptr	Ukazatel haldy
Table name	Název tabulky, kterou popisuje informace datového slovníku.

Mezipaměť distribuce dat

Optimalizátor dotazů používá statistiky rozdělení vytvořené příkazem UPDATE STATISTICS v režimu MEDIUM nebo HIGH k co nejméně náročnému určení plánu dotazů. Po prvním přístupu optimalizátoru ke statistikám rozdělení sloupce získá databázový server statistiky ze systémového katalogu **sysdistrib** na disku. Po tomto přístupu databázového serveru ke statistikám rozdělení je tato informace uložena do mezipaměti distribuce dat ve fyzické paměti.

Obrázek 4-3 zobrazuje přístup databázového serveru k mezipaměti distribuce dat pro více uživatelů. Po prvním přístupu optimalizátoru ke statistice rozdělení sloupce pro uživatele 1 uloží databázový server statistiku rozdělení do mezipaměti distribuce dat. Když optimalizátor určí plán dotazů pro uživatele 2, uživatele 3 a uživatele 4, kteří přistupují ke stejnému sloupci, databázový server nemusí číst informace o rozdělení dat tabulky z disku. Místo toho přečte statistiky rozdělení z mezipaměti distribuce dat ve sdílené paměti.



Obrázek 4-3. Mezipaměť distribuce dat

Databázový server uloží stránky systémového katalogu **sysdistrib**do společné oblasti vyrovnávací paměti stejně jako všechna ostatní data a indexové stránky. Mezipaměť distribuce dat poskytuje další výhody.

- Je uspořádána v efektivnějším formátu.
- Je uspořádána pro rychlé vyhledávání.
- Obchází zahlcení správy společné oblasti vyrovnávací paměti.
- Šetří místo ve společné oblasti vyrovnávací paměti pro stránky dat místo ukládání stránek systémového katalogu.
- Snižuje počet vstupně-výstupních operací tabulky systémového katalogu.

Konfigurace distribuce dat

Databázový server používá pro ukládání a umisťování informací v mezipaměti vyrovnávací paměti distribuce dat hashovací algoritmus. Konfigurační parametr DS_POOLSIZE řídí velikost mezipaměti distribuce dat a určuje celkový počet sloupcových rozdělení, které mohou být uloženy v mezipaměti distribuce dat. Chcete-li změnit počet sektorů v mezipaměti distribuce dat, použijte konfigurační parametr DS_HASHSIZE. Následující vzorec určuje počet sloupcových rozdělení, které lze uložit do jednoho sektoru.

```
Distributions_per_bucket = DS_POOLSIZE / DS_HASHSIZE
```

Chcete-li změnit počet rozdělení na sektor, změňte buď konfigurační parametr DS_POOLSIZE, nebo DS_HASHSIZE.

Například s výchozí hodnotou 127 parametru DS_POOLSIZE a výchozí hodnotou 31parametru DS_HASHSIZE můžete do mezipaměti distribuce dat uložit rozdělení až o 127 sloupcích. Mezipaměť má 31 sektorů hashovací tabulky a do každého sektoru lze uložit průměrně 4 položky.

Hodnoty, které nastavíte parametrům DS_HASHSIZE a DS_POOLSIZE závisí na následujících faktorech:

- Počet sloupců, u kterých provedete příkaz UPDATE STATISTICS v režimu HIGH nebo MEDIUM a očekáváte že budou použity v nejčastěji prováděných dotazech.

  Pokud neurčíte při spuštění příkazu UPDATE STATISTICS sloupec tabulky, databázový server vytvoří rozdělení pro všechny sloupce v tabulce.

  Jako vodítko při určování hodnot parametrů DS_HASHSIZE a DS_POOLSIZE můžete použít parametry DD_HASHSIZE a DD_HASHMAX. Parametry DD_HASHSIZE a DD_HASHMAX určují velikost mezipaměti datového slovníku, která uchovává informace a statistiky o tabulkách, ke kterým dotazy přistupují.

  U středních a velkých systémů můžete začít s následujícími hodnotami:

  – DD_HASHSIZE 503
  – DD_HASHMAX 4

– DS_HASHSIZE 503
– DS_POOLSIZE 2000

Monitorujte mezipaměti, abyste mohli parametry opravovat na základě aktuálního využití. Další informace o monitorování najdete v části “Monitorování mezipaměti distribuce dat” na stránce 4-23.

- Množství dostupné paměti

  Množství paměti potřebné k uložení rozdělení sloupce závisí na úrovni, na níž byl spuštěn příkaz UPDATE STATISTICS. Pro uložení rozdělení jednoho sloupce může být potřeba od 1 kB do 2 MB místa, v závislosti na tom, zda byl při spuštění příkazu UPDATE STATISTICS určen režim MEDIUM nebo HIGH nebo vyšší procentní rozlišení.

Pokud je mezipaměť dat příliš malá, mohou se objevit následující výkonnostní problémy:

- Databázový server použije hodnotu parametru DS_POOLSIZE k určení, kdy vymazat záznamy z mezipaměti distribuce dat. Pokud ale optimalizátor potřebuje vymazaná rozdělení pro jiný dotaz, databázový server musí znovu přistupovat k systémovému katalogu sysdistrib na disku. Další vstupně-výstupní operace a operace se společnou části vyrovnávacích pamětí potřebných pro přístup k systémovému katalogu sysdistrib na disku zvyšují celkový čas zpracování dotazu.

  Databázový server se pokusí udržet počet položek v mezipaměti distribuce dat na hodnotě parametru DS_POOLSIZE. Pokud celkový počet položek dosáhne vnitřní prahové hodnoty DS_POOLSIZE, databázový server použije ke smazání položek mechanizmus nejdéle nepoužité položky. Počet položek v sektoru hashovací tabulky může této hodnoty dosáhnout, ale databázový server nakonec stejně počet položek sníží, až se sníží nároky na paměť.
- Pokud je hodnota parametru DS_HASHSIZE malá a hodnota parametru DS_POOLSIZE velká, mohou být seznamy přetečení dlouhé a požadovat více času na vyhledávání v mezipaměti.

  K přetečení dojde, pokud sektor hashovací tabulky už obsahuje položku. Pokud více rozdělení hashuje do stejného sektoru, databázový server vytvoří seznam přetečení pro uložení a vyhledávání rozdělení.

  Pokud jsou hodnoty parametrů DS_HASHSIZE a DS_POOLSIZE přibližně stejné, seznam přetečení může být menší, nebo nemusí vůbec existovat, čímž může dojít k plýtvání pamětí. Každopádně je ale množství nepoužité paměti v takovém případě celkově bezvýznamné.

Monitorování mezipaměti distribuce dat

Chcete-li monitorovat velikost a využití mezipaměti distribuce dat, spusťte příkaz onstat -g dsc, nebo použijte volbu menu **ISA Performance -> Cache**. Pokud nastane některá z následujících situací, může být vhodné změnit hodnoty parametrů DS_HASHSIZE a DS_POOLSIZE:

- Pokud je mezipaměť distribuce dat většinu času plná a často používané sloupce nejsou vypsány v poli **distribution name**, zkuste zvětšit hodnoty parametrů DS_HASHSIZE a DS_POOLSIZE.
- Pokud je celkový počet položek mnohem menší než hodnota parametru DS_POOLSIZE, můžete hodnoty parametrů DS_HASHSIZE a DS_POOLSIZE snížit.

Obrázek 4-2 zobrazuje vzorový výstup příkazu **onstat -g dsc**.

```
onstat -g dsc



Distribution Cache:
    Number of lists             : 31
    DS_POOLSIZE                 : 127

Distribution Cache Entries:

list#id ref_cnt dropped? heap_ptr  distribution name
----------------------------------------------------------------

5     0       0       0   aa8f820  vjp_stores@gilroy:virginia.orders.order_num

12    0       0       0   aa90820  vjp_stores@gilroy:virginia.items.order_num

15    0       0       0   a7e9a38  vjp_stores@gilroy:virginia.customer.customer_num

19    0       0       0   aa3bc20  vjp_stores@gilroy:virginia.customer.lname

21    0       0       0   aa3cc20  vjp_stores@gilroy:virginia.orders.customer_num

28    0       0       0   aa91820  vjp_stores@gilroy:virginia.customer.company


Total number of distribution entries: 6.
    Number of entries in use    : 0
```

Obrázek 4-4. Výstup příkazu onstat -g dsc

Výstup příkazu **onstat -g dsc** má následující pole.

Pole	Popis
Number of Lists	Počet sektorů nebo seznamů určený parametrem DS_HASHSIZE.
DS_POOLSIZE	Počet rozdělení sloupců povolených v mezipaměti distribuce dat
Number of entries	Počet rozdělení sloupců aktuálně v mezipaměti distribuce dat
Number of entries in use	Počet aktuálně používaných rozdělení sloupců.
List #	Číslo sektoru hashovací tabulky
Id	Nepoužito
Ref cnt	Počet příkazů SQL aktuálně odkazujících na informace rozdělení dat tohoto sloupce v mezipaměti.
Dropped?	Označení, jestli bylo rozdělení sloupce označeno za neplatné klíčovým slovem DROP DISTRIBUTIONS příkazu UPDATE STATISTICS.
Heap ptr	Ukazatel haldy
Distribution name	Jméno tabulky, kterou popisuje informace rozdělení dat.

Mezipaměť příkazů SQL

Mezipaměť příkazů SQL ukládá analyzované a optimalizované příkazy SQL, takže uživatelé, kteří zároveň provádějí stejné příkazy SQL poznají následující zlepšení výkonu:

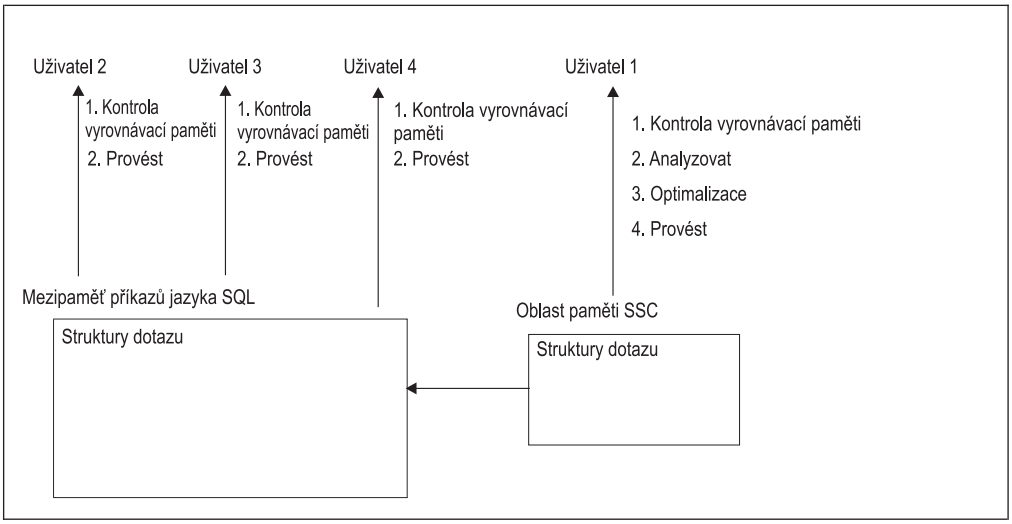
- Snížená doba odezvy, protože další příkazy už není třeba analyzovat a optimalizovat, jak ukazuje Obrázek 4-5.

- Snížené využití paměti, protože databázový server sdílí struktury dat dotazů mezi jednotlivými uživateli.

Další informace o vlivu mezipaměti příkazů SQL na výkon jednotlivých dotazů najdete v části “Mezipaměť příkazů SQL” na stránce 13-27.

Obrázek 4-5 zobrazuje jak databázový server přistupuje do mezipaměti příkazů SQL při více zároveň pracujících uživatelích.

- Když databázový server provede příkaz SQL poprvé pro uživatele 1, zkontroluje, zda není stejný příkaz uložen v mezipaměti příkazů SQL. Pokud není v mezipaměti, dabázový server analyzuje příkaz, určí optimální plán dotazu a příkaz provede.
- Když bude chtít uživatel 2 provést stejný příkaz SQL, databázový server najde příkaz v mezipaměti příkazů SQL a nemusí jej už analyzovat a optimalozovat.
- Podobně, pokud chtějí uživatelé 3 a 4 provést stejný příkaz SQL, databázový server už je nemusí znovu analyzovat a optimalizovat Místo toho použije informace plán dotazů z mezipaměti příkazů SQL ve fyzické paměti.



Obrázek 4-5. ¨ Akce databázového serveru při použití mezipaměti pro příkazy SQL.

Připravené příkazy a mezipaměť příkazů

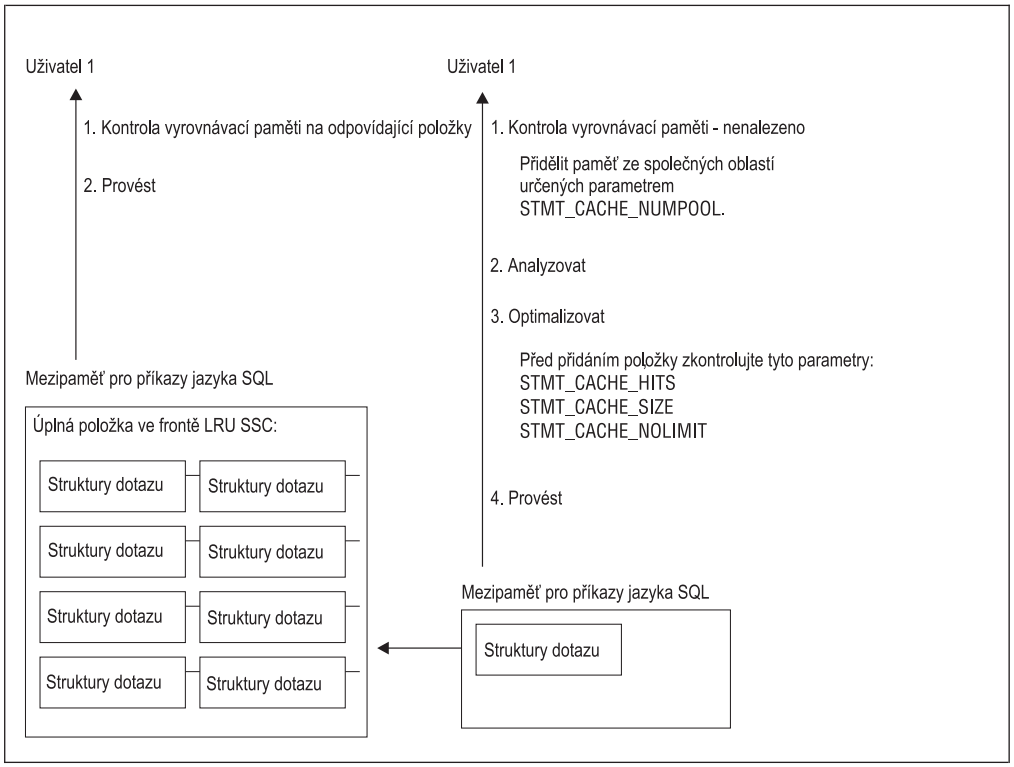
Připravené příkazy jsou pro jednoduchou relaci automaticky ukládány do mezipaměti. To znamená, že pokud je vícekrát proveden připravený příkaz, nebo pokud je vícekrát otevřen jednoduchý kurzor, relace použije stejný připravený plán dotazu. Pokud relace připraví příkaz a pak jej vícekrát provede, nemá na výsledný výkon vliv mezipaměť příkazů SQL, protože příkaz je optimalizován už během přípravy.

Každopádně, pokud si jiná relace připraví stejný příkaz, nebo pokud první relace připraví příkaz několikrát, poskytuje mezipaměť příkazů přímou výkonnostní výhodu, protože databázový server vypočítává plán dotazů pouze jednou. Původní relace samozřejmě díky mezipaměti získá také výhodu, přestože příkaz připravila, protože ostatní relace zabírají méně paměti a databázový server nemusí vykonávat tolik práce pro ostatní relace.

Konfigurace mezipaměti příkazů SQL

Hodnota konfiguračního parametru STMT_CACHE povoluje nebo zakazuje mezipaměť příkazů SQL, jak popisuje část “Povolení mezipaměti příkazů SQL” na stránce 13-29.

Obrázek 4-6 zobrazuje, jak databázový server používá hodnoty vhodných konfiguračních parametrů pro mezipaměť příkazů SQL. Další vysvětlení následují pod obrázkem.



Obrázek 4-6. Konfigurační parametry, které ovlivňují mezipaměť příkazů SQL.

Když databázový server používá pro uživatele mezipaměť příkazů SQL, znamená to, že databázový server provádí následující akce:

- Nejdříve zkontroluje mezipaměť příkazů SQL, zda se v ní nenachází stejný příkaz SQL, jaký chce uživatel provést.
- Pokud příkaz SQL odpovídá položce, provede příkaz za použití paměťové struktury dotazu v mezipaměti příkazů SQL (uživatel 2, Obrázek 4-6)
- Pokud příkaz SQL neodpovídá položce, databázový server zkontroluje, zda je příkaz vhodný pro uložení do mezipaměti.

  Další informace o vhodnosti příkazu SQL pro uložení do mezipaměti najdete v příkazu SET STATEMENT CACHE v části *IBM Informix Guide to SQL: Syntax*.
- Pokud je příkaz SQL vhodný, je uložen do mezipaměti k dalšímu zpracování.

Následující parametry ovlivňují, zda databázový server vloží nebo nevloží příkaz SQL do mezipaměti (uživatel 1, Obrázek 4-6 na stránce 4-26):

- Hodnota parametru STMT_CACHE_HITS určuje kolikrát je příkaz proveden s položkou v mezipaměti (odkazována jako *počet přístupů*). Databázový server v závislosti na počtu přístupů vloží jednu z následujících položek:
  - Pokud je hodnota parametru STMT_CACHE_HITS 0, bude do mezipaměti vložena úplná položka obsahující text příkazu SQL a strukturu paměti dotazu.
  - Pokud hodnota parametru STMT_CACHE_HITS není 0 a příkaz v mezipaměti neexistuje je vložena pouze klíčová část položky obsahující text příkazu SQL. Další provádění příkazu SQL zvyšují počet přístupů.
  - Pokud je hodnota parametru STMT_CACHE_HITS stejná jako počet přístupů klíčové části položky, je přidána struktura paměti dotazu a je tak vytvořena úplná položka.

- Parametr STMT_CACHE_SIZE určuje velikost mezipaměti příkazů SQL a parametr STMT_CACHE_NOLIMIT určuje, zda bude mezipaměť omezena na hodnotu parametru STMT_CACHE_SIZE. Pokud neurčíte hodnotu parametru STMT_CACHE_SIZE, výchozí hodnota je 524288 (512 * 1024) bajtů.

  Výchozí hodnota parametru STMT_CACHE_NOLIMIT je 1, což znamená, že databázový server bude vkládat položky do mezipaměti příkazů SQL i přesto, že celkové množství paměti může přesáhnout hodnotu parametru STMT_CACHE_SIZE.

  Pokud je hodnota parametru STMT_CACHE_NOLIMIT nastavena na 0, databázový server vloží příkaz SQL do mezipaměti, pouze pokud velikost mezipaměti nepřesáhne limit paměti.

Následující část poskytuje více podrobností o tom, jak jednotlivé konfigurační parametry ovlivňují mezipaměť příkazů SQL a důvody, proč by mohlo být vhodné změnit jejich výchozí hodnoty.

- STMT_CACHE_HITS
- STMT_CACHE_SIZE
- STMT_CACHE_NOLIMIT
- STMT_CACHE_NUMPOOL

Monitorování a ladění mezipaměti příkazů SQL

Je možné monitorovat a vyladit různé charakteristiky mezipaměti příkazů SQL. Následující tabulka zobrazuje nástroje, které je možné použít k monitorování různých charakteristik. Program ISA používá ke zobrazení informací o mezipaměti SQL výstup, který generují následující volby příkazu **onstat**. Klepnutím na tlačítko **Refresh** spustíte příkaz **onstat** znovu a zobrazíte aktuální informace.

Monitorovaná charakteristika SSC	Výběr v programu ISA	Zobrazí výstup příkazu	Další informace
Kolikrát jsou příkazy přečteny z mezipaměti.	**Performance > Cache > Statement Cache**	**onstat -g ssc** **onstat -g ssc all**	"Počet provedení příkazu SQL" na stránce 4-27
Velikost mezipaměti příkazů SQL	**Performance > Cache > Statement Cache**	**onstat -g ssc** **onstat -g ssc all**	"Monitorování a ladění velikosti mezipaměti příkazů SQL" na stránce 4-29
Množství použité paměti	**Performance > Cache > Statement Cache**	**onstat -g ssc** **onstat -g ssc all**	"Omezení paměti a velikost" na stránce 4-31
Použití společné oblasti mezipaměti příkazů SQL	**Performance > Locks > Latches**	**onstat -g spi**	"Vícenásobná společná oblast mezipamětí příkazů SQL" na stránce 4-32

Počet provedení příkazu SQL

Pokud je povolena mezipaměť příkazů SQL, databázový server jako výchozí nastavení okamžitě ukládá vhodný příkaz SQL a jeho struktury paměti do mezipaměti příkazů SQL. Pokud je v hodnotě zatížení nepřiměřený počet dotazů ad hoc, pomocí konfiguračního parametru STMT_CACHE_HITS určete počet, kolikrát musí být příkaz SQL proveden, než jej databázový server uloží jako úplný dotaz do mezipaměti příkazů SQL.

Pokud je hodnota konfiguračního parametru STMT_CACHE_HITS větší než 0 a počet provedení příkazu SQL menší než hodnota parametru STMT_CACHE_HITS, databázový server vloží do mezipaměti pouze klíčové části položek. Tato specifikace zabraňuje strukturám nesdílené paměti zabírat místo v mezipaměti příkazů, čímž zůstává více místa pro příkazy SQL, které aplikace často používají.

Monitorováním počtu přístupů do mezipaměti příkazů SQL zjistíte, jestli sytém využívá tuto paměť efektivně. Následující části popisují způsoby, jak monitorovat počet přístupů do mezipaměti příkazů SQL.

Použití příkazu onstat -g ssc k monitorování počtu přístupů na SSC: Příkaz **onstat -g ssc** zobrazí úplné položky v mezipaměti příkazů SQL. Obrázek 4-7 zobrazuje výstup příkazu **onstat -g ssc**.

```
onstat -g ssc

Statement Cache Summary:
#lrus   currsize  maxsize   Poolsize   #hits     nolimit
4       49456     524288    57344      0         1

Statement Cache Entries:

lru hash ref_cnt  hits  flag heap_ptr      database           user
---------------- ----- ------------------------------------------------------
  0  153       0      0    -F  a7e4690      vjp_stores         virginia
  SELECT * FROM customer, orders
    WHERE customer.customer_num = orders.customer_num
     AND order_date > "01/01/07"

  1  259       0      0    -F  aa58c20      vjp_stores         virginia
  SELECT * FROM customer, orders
    WHERE customer.customer_num = orders.customer_num
     AND order_date > "01/01/2007"

  2  232       0      1    DF  aa3d020      vjp_stores         virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
     AND O.order_num = I.order_num

  3  232       1      1    -F  aa8b020      vjp_stores         virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num



    Total number of entries: 4.
```

Obrázek 4-7. Výstup příkazu onstat -g ssc

Tip: Příkaz **onstat -g ssc** je ekvivalentní příkazu **onstat -g cac stmt** verze 9.2. Příkaz **onstat -g cac stmt** v aktuální verzi databázového serveru zobrazuje stejné sloupce jako příkaz **onstat -g ssc**.

Chcete-li monitorovat, kolikrát databázový server přečte příkaz SQL z mezipaměti , sledujte následující sloupce výstupu:

- V části Statement Cache Summary výstupu příkazu **onstat -g ssc** představuje sloupec **#hits** hodnotu konfiguračního parametru SQL_STMT_HITS.

  Obrázek 4-7 na stránce 4-28 zobrazuje sloupec **#hits** v části výstupu Statement Cache Summary hodnotu 0, což je výchozí hodnota konfiguračního parametru STMT_CACHE_HITS.

  Důležité: Databázový server používá položky z mezipamětipříkazů SQL pouze v případě, že jsou příkazy zcela totožné. První dvě položky, které znázorňuje

Obrázek 4-7 na stránce 4-28, nejsou zcela stejné, protože každá z nich obsahuje jinou literální hodnotu ve filtru **order_date**.

- V části Statement Cache Entries výstupu příkazu **onstat -g ssc** udává sloupec hits, kolikrát databázový server provedl každý příkaz SQL z mezipaměti. Jinými slovy, číslo v tomto sloupci udává, kolikrát databázový server použil paměťové struktury z mezipaměti, aniž by bylo nutné příkazy znovu vytvářet, analyzovat a optimalizovat.

  Po prvním vložení příkazu do mezipaměti je hodnota sloupce **hits** rovna 0.

  – První dva příkazy SQL, které znázorňuje Obrázek 4-7, mají ve sloupci **hits** hodnotu 0, což znamená, že tyto příkazy byly vloženy do mezipaměti, ale ještě z ní nebyly provedeny.

  – Poslední dva příkazy SQL, které znázorňuje Obrázek 4-7, mají ve sloupci **hits** hodnotu 1, což znamená, že tyto příkazy byly z mezipaměti jednou provedeny.

  Hodnota **hits** pro jednotlivé položky ukazuje, jak je využíváno sdílení paměťových struktur. Vyšší hodnoty ve sloupci **hits** ukazují, že mezipaměť příkazů SQL je při zvyšování výkonu a zlepšování využití paměti užitečná.

Úplný popis výstupních polí příkazu **onstat -g ssc** najdete v části "Popis výstupu voleb příkazu onstat pro mezipaměť příkazů SQL" na stránce 4-34.

Použití příkazu onstat -g ssc all: Pomocí příkazu **onstat -g ssc all** můžete zjistit, kolik nesdílených položek existuje v mezipaměti. Příkaz **onstat -g ssc all** zobrazí navíc kromě úplných položek i klíčové části položek v mezipaměti příkazů SQL.

Chcete-li zjistit, kolik nesdílených položek je celkem v mezipaměti vyrovnávací paměti uloženo:

1. Porovnejte výstup příkazu **onstat -g ssc all** s výstupem příkazu **onstat -g ssc**.
2. Pokud rozdíl mezi těmito dvěma výstupy ukazuje, že v mezipaměti příkazů SQL je uloženo množství nesdílených položek, zvyšte hodnotu konfiguračního parametru STMT_CACHE_HITS, čímž povolíte uložení více sdílených příkazů a snížíte zahlcení mezipaměti příkazů SQL.

Ke změně hodnoty parametru STMT_CACHE_HITS můžete použít jeden z následujících postupů:

- Aktualizovat soubor ONCONFIG a určit hodnotu konfiguračního parametru STMT_CACHE_HITS. Uplatnění změn vyžaduje, abyste restartovali databázový server.

  K aktualizaci souboru ONCONFIG můžete použít jeden z následujících způsobů:

  – V programu ISA přejděte na stránku **Configuration** a přidejte na tuto stránku parametr STMT_CACHE_HITS. Následně přejděte na stránku **Mode** a restartujte databázový server.

  – Pomocí textového editoru upravte soubor ONCONFIG. Následně databázový server zastavte příkazem **onmode -ky** a restartujte jej příkazem **oninit**.

- Použít příkaz **onmode -W** nebo stránku **mode** programu ISA a zvýšit hodnotu parametru STMT_CACHE_HITS dynamicky za běhu databázového serveru.

  ```
  onmode -W STM_CACHE_HITS 2
  ```

  Pokud databázový server restartujete, hodnota parametru se vrátí na původní hodnotu definovanou v souboru ONCONFIG. Z toho důvodu, pokud chcete, aby nastavení zůstalo zachováno i po následném restartování databázového serveru, upravte soubor ONCONFIG.

Monitorování a ladění velikosti mezipaměti příkazů SQL

Pokud je mezipaměť příkazů SQL malá, může dojít k následujícím výkonnostním problémům:

- Často prováděné příkazy SQL nejsou uloženy v mezipaměti.

Nejčastěji využívané příkazy by měly zůstat v mezipaměti příkazů SQL. Pokud není mezipaměť příkazů SQL dostatečně velká, databázový server nemusí mít dostatek místa pro uložení těchto příkazů do mezipaměti. Při dalším provádění musí databázový server příkazy znovu analyzovat, optimalizovat a znovu se je pokusit uložit do mezipaměti příkazů SQL. Zkuste zvýšit hodnotu parametru STMT_CACHE_SIZE.

- Databázovému serveru zabere dlouhou dobu vymazání příkazu SQL z mezipaměti.

  Databázový server pomocí prahové hodnoty (70 procent hodnoty parametru STMT_CACHE_SIZE) určuje, kdy odebrat položky z mezipaměti příkazů SQL a zkouší tak zabránit tomu, aby byla příkazům SQL přidělena příliš velká část paměti. Pokud nová položka způsobí, že velikost mezipaměti příkazů SQL přesáhne prahovou hodnotu, databázový server před vložením nové položky odstraní nejdéle nepoužívané položky (pokud se zrovna nepoužívají).

  Pokud ale následující dotaz potřebuje odstraněné paměťové struktury, databázový server musí příkaz SQL znovu analyzovat a optimalizovat. Tento čas potřebný k opětovnému vytvoření paměťových struktur se promítne do celkového času potřebného ke zpracování dotazu.

Velikost mezipaměti příkazů SQL ve fyzické paměti můžete nastavit konfiguračním parametrem STMT_CACHE_SIZE. Hodnota parametru je velikost paměti v kB. Pokud není hodnota parametru STMT_CACHE_SIZE nastavena, výchozí hodnotou je 512 kB.

Výstup příkazu **onstat -g ssc** zobrazuje hodnotu parametru STMT_CACHE_SIZE ve sloupci **maxsize**. Obrázek 4-7 ukazuje ve sloupci **maxsize** hodnotu 524288, což je výchozí hodnota (512 * 1024 = 524288).

Pomocí příkazů **onstat -g ssc** a **onstat -g ssc all** můžete monitorovat efektivitu velikosti mezipaměti příkazů SQL. Pokud nejsou v mezipaměti příkazů SQL zobrazeny položky, které aplikace nejčastěji používají, je možné, že je mezipaměť příkazů SQL příliš malá nebo ji zabírá velké množství nesdílených příkazů SQL. Následující části popisují, jak tyto situace určit.

Změna velikosti mezipaměti příkazů SQL, protože je příliš malá: Pokud je mezipaměť příkazů SQL příliš malá, můžete její velikost změnit.

Chcete-li zjistit, zda je mezipaměť příkazů SQL příliš malá:

1. Spusťte příkaz **onstat -g ssc all**.
2. Sledujte hodnoty v následujících sloupcích části “Statement Cache Entries” výstupu příkazu **onstat -g ssc all** :
   - Sloupec **flags** zobrazuje aktuální stav příkazu SQL v mezipaměti.

     Hodnota F na druhé pozici ukazuje, že je příkaz uložen do mezipaměti celý.

     Hodnota - na druhé pozici ukazuje, že je do mezipaměti uložen pouze text příkazu (klíčová část položky). Položky s hodnotou - na druhé pozici se zobrazí ve výstupu příkazu **onstat -g ssc all** , ale ne ve výstupu příkazu **onstat -g ssc**.
   - Sloupec **hits** ukazuje, kolikrát byl příkaz SQL proveden. První provedení, při kterém byl uložen do mezipaměti, se nepočítá.

   Pokud nejsou zobrazeny úplné položky příkazů, které aplikace nejčastěji využívají a hodnoty ve sloupci **hits** položek v mezipaměti jsou vysoké, pak je mezipaměť příkazů SQL příliš malá.

Chcete-li změnit velikost mezipaměti příkazů SQL:

1. Aktualizujte hodnotu konfiguračního parametru STMT_CACHE_SIZE.
2. Restartujte databázový server, aby se změny projevily.

Příliš mnoho jednorázových dotazů v mezipaměti příkazů SQL: Když databázový server uloží do mezipaměti dotazy, které jsou použity pouze jednou, může tím nahradit dotazy, které ostatní aplikace používají často.

Podívejte se na hodnoty v následujících sloupcích části Statement Cache Entries výstupu příkazu **onstat -g ssc all** . Pokud je zobrazena spousta položek, které mají obě následujíccí hodnoty, je v mezipaměti příliš mnoho nesdílených příkazů SQL:

- sloupec **flags** s hodnotou F na druhé pozici

  Hodnota F na druhé pozici ukazuje, že je příkaz uložen do mezipaměti celý.
- sloupec **hits** s hodnotou 0 nebo 1

  Sloupec **hits** ukazuje, kolikrát byl příkaz SQL proveden. První provedení, při kterém byl uložen do mezipaměti, se nepočítá.

Zvyšte hodntou konfiguračního parametru STMT_CACHE_HITS, čímž předejdete ukládání úplných položek nesdílených příkazů SQL do mezipaměti. Další informace o změně hodnoty konfiguračního parametru STMT_CACHE_HITS najdete v části “Počet provedení příkazu SQL” na stránce 4-27.

Omezení paměti a velikost

Ačkoli se server pokouší položky z mezipaměti příkazů SQL odstraňovat, někdy nelze položky odebrat, protože jsou právě používány. V tomto případě může velikost mezipaměti příkazů SQL přesáhnout hodnotu parametru STMT_CACHE_SIZE.

Výchozí hodnota konfiguračního parametru STMT_CACHE_NOLIMIT je 1, což znamená, že databázový server vloží příkaz i přesto, že aktuální velikost mezipaměti může být větši než hodnota parametru STMT_CACHE_SIZE.

Pokud je hodnota konfiguračního parametru STMT_CACHE_NOLIMIT 0, databázový server nevloží další ani vhodnou ani pouze klíčovou položku, pokud by měla celková velikost mezipaměti příkazů SQL přesáhnout hodnotu parametru STMT_CACHE_SIZE.

Aktuální velikost mezipaměti příkazů SQL můžete monitorovat příkazem **onstat -g ssc** . Sledujte hodnoty v následujících sloupcích části “Statement Cache Entries” výstupu příkazu**onstat -g ssc**:

- Sloupec **currsize** zobrazuje počet bajtů aktuálně přidělených v mezipaměti příkazů SQL.

  Obrázek 4-7 na stránce 4-28 znázorňuje v sloupci **currsize** hodnotu 11264.
- Sloupec **maxsize** zobrazuje hodnotu parametru STMT_CACHE_SIZE.

  Obrázek 4-7 znázorňuje v sloupci **maxsize** hodnotu 524288, což je výchozí hodnota (512 * 1024 = 524288).

Pokud je mezipaměť příkazů SQL plná a uživatelé aktuálně provádějí všechny příkazy z ní, může jakýkoli další příkaz SQL způsobit, že se velikost mezipaměti příkazů SQL zvětší nad hodnotu parametru STMT_CACHE_SIZE. Pokud už databázový server nevyužívá příkaz SQL z mezipaměti příkazů SQL, začne uvolňovat místo v mezipaměti, dokud se její velikost zase nesníží pod prahovou hodnotu parametru STMT_CACHE_SIZE. V každém případě pokud tisíce uživatelů provádějí souběžně dotazy ad hoc, může velikost mezipaměti příkazů SQL velmi rychle vzrůst, dříve než budou některé příkazy odstraněny. V takovýchto případech proveďte jednu z následujících akcí:

- Nastavte hodnotu parametru STMT_CACHE_NOLIMIT na 0, čímž zamezíte vkládání dalších příkazů, pokud velikost mezipaměti překročí hodnotu parametru STMT_CACHE_SIZE.
- Nastavte parametr STMT_CACHE_HITS na hodnotu větší než 0, čímž zamezíte ukládání nesdílených příkazů SQL do mezipaměti.

Hodnotu parametru STMT_CACHE_NOLIMIT můžete změnit jedním z následujících způsobů:

- Aktualizujte soubor ONCONFIG a upravte hodnotu konfiguračního parametru STMT_CACHE_NOLIMIT. Uplatnění změn vyžaduje, abyste restartovali databázový server.
- Pomocí příkazu **onmode -W** můžete změnit hodnotu konfiguračního parametru STMT_CACHE_NOLIMIT dynamicky za běhu databázového serveru.

  ```
  onmode -W STMT_CACHE_NOLIMIT 0
  ```

  Pokud databázový server restartujete, hodnota parametru se vrátí na původní hodnotu definovanou v souboru ONCONFIG. Z toho důvodu, pokud chcete, aby nastavení zůstalo zachováno i po následném restartování databázového serveru, upravte soubor ONCONFIG.

Vícenásobná společná oblast mezipamětí příkazů SQL

Pokud je mezipaměť příkazů SQL povolena, databázový server dostane přidělenu paměť z jedné společné oblasti (jako výchozí nastavení) pro struktur dotazu v následujících situacích:

- Když databázový server nenajde v mezipaměti dopovídající dotaz.
- Pokud databázový server najde odpovídající v mezipaměti klíčovou část položky a počet přístupů odpovídá hodnotě konfiguračního parametru STMT_CACHE_HITS.

Tato část společné oblasti se může při zvýšení počtu uživatelů stát kritickým místem. Konfigurační parametr STMT_CACHE_NUMPOOL umožňuje konfigurovat vícenásobné společné oblasti **sscpools**.

Společné oblasti mezipamětí příkazů SQL můžete monitorovat a zjistit následující situace:

- Počet společných oblastí sscpools je pro aktuální zatížení dostatečný.
- Velikost nebo limit mezipaměti příkazů SQL nezpůsobuje nadměrnou správu paměti.

Počet společných oblastí mezipamětí příkazů SQL: Pokud je mezipaměť příkazů SQL povolena, databázový server přidělí paměť ze společné oblasti sscpool pro nespojené příkazy SQL. Výchozí hodnota konfiguračního parametru STMT_CACHE_NUMPOOL je 1. Při zvýšení počtu uživatelů se může tato společná oblast sscpool stát úzkým hrdlem. (počet zámků longspin ve společné oblasti sscpool určuje, zda společná oblast sscpool je nebo není úzkým hrdlem.)

Pomocí příkazu **onstat -g spi** můžete monitorovat počet zámků longspin ve společné oblasti sscpool. Příkaz **onstat -g spi** zobrazí seznam zdrojů v systému, u kterých se muselo čekat na získání zámku ke zdroji. Během čekání jednotkový proces probíhá ve smyčce a snaží se získat systémové prostředky. Výstup příkazu **onstat -g spi** zobrazí počet čekacích cyklů (sloupec **Num Waits**), které proběhly, než byly prostředky přiděleny a celkový počet cyklů (sloupec **Num Loops**). Výstup příkazu **onstat -g spi** zobrazuje pouze prostředky, které prošly aspoň jedním čekacím cyklem.

Obrázek 4-8 zobrazuje část vzorového výstupu příkazu **onstat -g spi**. Obrázek 4-8 ukazuje, že pro společné části sscpool neproběhly žádné čekací cykly (sloupec **Name** neobsahuje žádné společné části sscpools).

```
Spin locks with waits:
Num Waits   Num Loops   Avg Loop/Wait    Name
34477       387761      11.25            mtcb sleeping_lock
312         10205       32.71            mtcb vproc_list_lock
```

Obrázek 4-8. Výstup příkazu onstat -g spi.

Pokud je zobrazeno nepřiměřené číslo zámků longspin (čekacích cyklů) (sloupec **Num Loops**) společné oblasti sscpool, zvyšte pro zvýšení výkonu systému hodntou počtu společných oblastí sscpools v konfiguračním parametru STMT_CACHE_NUMPOOL.

Velikost společné oblasti mezipamětí příkazů SQL a aktuální velikost mezipaměti: Pomocí příkazu **onstat -g ssc pool** můžete monitorovat využití každé společné oblasti mezipaměti příkazů SQL. Příkaz **onstat -g ssc pool** zobrazí velikost každé společné oblasti. Příkaz **onstat -g ssc** zobrazuje kumulativní velikost mezipaměti příkazů SQL ve sloupci **currsize**. Tato aktuální velikost je velikostí fyzické paměti přidělené ze společných oblastí sscpool příkazy vloženými do mezipaměti. Protože ne všechny příkazy, kterým je přidělena paměť ze společných oblastí sscpool, jsou uloženy do mezipaměti, aktuální velikost mezipaměti může být menší než celková velikost společných oblastí sscpool. Běžně celková velikost společných oblastí sscpool nepřekročí hodnotu parametru STMT_CACHE_SIZE.

Obrázek 4-9 zobrazuje vzorový výstup příkazu **onstat -g ssc pool**.

```
onstat -g ssc pool


Pool Summary:
name         class addr     totalsize freesize #allocfrag #freefrag
sscpool0     V     a7e4020  57344      2352     52         7

Blkpool Summary:
name         class addr     size      #blks
```

Obrázek 4-9. Výstup příkazu onstat -g ssc pool.

Část Pool Summary výstupu příkazu **onstat -g ssc pool** zobrazuje pro každou společnou oblast v mezipaměti následující informace**.**

Sloupec	**Popis**
name	Jméno společné oblasti sscpool.
class	Segment sdílené paměti, ve kterém byla společná oblast vytvořena. U společných oblastí sscpool je tato hodnota vždy “V”, což značí virtuální část sdílené paměti.
addr	Adresa sdílené paměti struktury společné části SSC pool.
totalsize	Celková velikost této společné části SSC pool v bajtech.
freesize	Počet volných bajtů v této společné části SSC pool.
#allocfrag	Počet bezprostředně po sobě následujících oblastí paměti, kterým je v této společné oblasti sscpool přidělena paměť.
#freefrag	Počet bezprostředně po sobě následujících oblastí, které nejsou v této společné části SSC pool využity.

Část Blkpool Summary výstupu příkazu **onstat -g ssc pool** zobrazuje pro každou společnou oblast v mezipaměti následující informace **.**

Sloupec	**Popis**

name	Jméno společné oblasti sscpool.
class	Segment sdílené paměti, ve kterém byla společná oblast vytvořena. U společných oblastí sscpool je tato hodnota vždy “V”, což značí virtuální část sdílené paměti.
addr	Adresa sdílené paměti struktury společné části SSC pool.
size	Celková velikost této společné části SSC pool v bajtech.
#blks	Počet osmikilobajtových bloků ze kterých jsou vytvořeny všechny společné oblasti SSC pools.

Popis výstupu voleb příkazu onstat pro mezipaměť příkazů SQL

Příkaz **onstat -g ssc** zobrazí následující souhrnné informace o mezipaměti příkazů SQL**.**

Sloupec	Popis
#lrus	Počet dotazů LRU. Vícenásobné dotazy LRU usnadňují současné vyhledávání a vkládání položek mezipaměti.
currsize	Počet bajtů aktuálně přidělených položkám mezipaměti příkazů SQL.
maxsize	Počet bajtů určených v konfiguračním parametru STMT_CACHE_SIZE.
poolsize	Kumulativní počet bajtů všech společných oblastí v mezipaměti příkazů SQL. Pomocí příkazu **onstat -g ssc pool** můžete monitorovat využití jednotlivých společných oblastí.
#hits	Aktuální nastavení hodnoty konfiguračního parametru STMT_CACHE_HITS, který určuje počet, kolikrát je příkaz proveden než je uložen do mezipaměti.
nolimit	Nastavení Current konfiguračního parametru STMT_CACHE_NOLIMIT

Příkaz **onstat -g ssc** zobrazí pro každou úplnou položku z mezipaměti následující informace**.** Příkaz **onstat -g ssc all** tyto informace zobrazí jak pro úplné položky, tak pro klíčové části položek.

Sloupec	Popis
lru	Indentifikátor LRU.
hash	Identifikátor sektoru hashovací tabulky.
ref_cnt	Počet relací aktuálně využívajících tento příkaz.
hits	Počet, kolikrát byl dotaz přečten z mezipaměti (nepočítá se první přečtení, kdy byl příkaz uložen do mezipaměti).
flags	Příznakové kódy pro pozici 1: **D** Ukazuje, že příkaz byl vypuštěn a již není aktuální. Příkaz v mezipaměti může být vypuštěn (nebude více použit), pokud se změnila některá z jeho závislostí. Například když spustíte v tabulce příkaz UPDATE STATISTICS, statistické údaje optimalizátoru se mohou změnit a tím se může změnit i plán dotazů pro příkazy v mezipaměti příkazů SQL na zastaralý. V tomto případě označí databázový server při dalším pokusu o použití tento příkaz za vypuštěný. **-** Ukazuje, že příkaz nebyl vypuštěn a je stále aktuální. Příznakové kódy pro pozici 2: **F** Ukazuje, že položka byla uložena do mezipamětiúplně a obsahuje paměťové struktury dotazu.

-	Ukazuje, že příkaz nebyl uložen do mezipaměti jako úplný. Příkaz není do mezipaměti vložen úplně, pokud počet, kolikrát byl proveden, je nižší než hodnota konfiguračního parametru STMT_CACHE_HITS. Položky s hodnotou - na druhé pozici se zobrazí ve výstupu příkazu **onstat -g ssc all**, ale ne ve výstupu příkazu **onstat -g ssc**.

heap_ptr	Ukazatel na haldu přiřazenou k příkazu.
database	Databáze, proti které je příkaz SQL prováděn.
user	Uživatel provádějící příkaz SQL.
statement	Text příkazu, jak bude použit pro zkoušku shody.

Paměť relace

Databázový server používá virtuální část sdílené paměti především pro uživatelské relace. Většina paměti, která je řuživateli přidělena je použita na příkazy SQL. Množství použité paměti se může lišit příkaz od příkazu.

Pomocí následujících voleb obslužných programů můžete zjistit, které relace a připravené příkazy SQL využívají nejvíce paměti:

- **onstat -g mem**
- **onstat -g stm**

Příkaz **onstat -g mem** zobrazuje využití paměti pro všechny relace. Relaci, která zabírá nejvíce paměti najdete podle sloupců **totalsize** a **freesize** výstupu příkazu. Obrázek 4-11 zobrazuje vzorový výstup příkazu **onstat -g mem**. Tento vzorový výstup zobrazuje využití paměti tří uživatelských relací s hodnotami 14, 16 a 17 ve sloupci **names**.

```
onstat -g mem

Pool Summary:
name         class addr     totalsize freesize #allocfrag #freefrag
...
14           V     a974020  45056     11960    99         10
16           V     a9ea020  90112     10608    159        5
17           V     a973020  45056     11304    97         13
...
Blkpool Summary:
name         class addr     size      #blks
mt           V     a235688  798720    19
global       V     a232800  0         0
```

Obrázek 4-10. Výstup příkazu onstat -g mem

Chcete-li zobrazit přidělenou každému připravenému příkazu, použijte příkaz **onstat -g stm**. Obrázek 4-11 zobrazuje výstup příkazu **onstat -g stm**.

```
onstat -g stm


session   25 ------------------------------------------------
 sdblock  heapsz  statement ('*' = Open cursor)
 d36b018    9216  select sum(i) from t where i between -1 and ?
 d378018    6240  *select tabname from systables where tabid=7
 d36b114    8400  <SPL statement>
```

Obrázek 4-11. Výstup příkazu onstat -g stm

Množství paměti použité příkazem zobrazuje sloupec **heapsz** výstupu na obrázku Obrázek 4-11. Hvězdička (*) předchází příkazu v případě, že je na něm otevřen kurzor. Výstup nezobrazuje jednotlivé příkazy SQL v SPL rutině.

Chcete-li zobrazit paměť pouze pro jednu relaci, určete ID relace v příkazu **onstat -g stm**. Příklad naleznete v části “Příkazy onstat -g mem a onstat -g stm” na stránce 13-36.

Vyrovnávací paměti replikace dat a využití paměti

Replikace dat vyžaduje dvě instance databázového serveru, primární a sekundární, které jsou spuštěny na dvou různých počítačích. Pokud na svém databázovém serveru implementujete replikaci dat, tento udržuje záznamy logického protokolu ve vyrovnávací paměti replikace dat do té doby, než je pošle na sekundární databázový server. Vyrovnávací paměť replikace dat je vždy stejně velká jako vyrovnávací paměť logického protokolu.

Zámky paměti

Databázový server používá zámky pro řízení přístupu ke strukturám sdílené paměti, jako jsou společné oblasti vyrovnávacích pamětí nebo společné oblasti paměti pro mezipaměť příkazů SQL.

Můžete získat statistické údaje o použití zámků latch a informace o určitých zámcích latch. Tyto statistické údaje udávají měřítko aktivity systému. Statistické údaje obsahují i počet, kolikrát musel na získání zámku jednotkový proces čekat. Velký počet čekání na zámek typicky vyúsťuje ve velké množství zpracovávající aktivity, při které databázový server protokoluje většinu transakcí.

Informace o specifických zámcích obsahují seznam všech zámků, které v držení jednotkových procesů a všech jednotkových procesů, které čekají na zámky. Díky této informaci můžete vyhledat všechny možné kolize zdrojů.

Jako administrátor databáze nemůžete konfigurovat nebo ladit počet zámků. Ale můžete pro snížení počtu čekání na zámek zvětšit velikost paměti, do které databázový server umisťuje zámky. Například můžete vyladit počet společných oblastí mezipaměti příkazů SQL nebo počet LRU front mezipaměti příkazů SQL. Další informace najdete v části “Vícenásobná společná oblast mezipamětí příkazů SQL” na stránce 4-32.

Upozornění: Nikdy neodstraňujte databázový proces, který drží zámek. Pokud tak učiníte, databázový server okamžitě způsobí přerušení.

Monitorování zámků za použití obslužných programů příkazového řádku

Následující obslužné programy příkazového řádku můžete použít k získání informací o zámcích.

onstat -p

Provedením příkazu **onstat -p** získáte hodnoty pole **lchwaits**. V tomto poli jsou uloženy počty, kolikrát musel jednotkový proces čekat na zámek sdílené paměti.

Obrázek 4-12 zobrazuje výtah ze vzorového výstupu příkazu **onstat -p**, který zobrazuje pole lchwaits.

```
...
ixda-RA  idx-RA   da-RA    RA-pgsused lchwaits
5        0        204      148        12
...
```

Obrázek 4-12. Výstup příkazu onstat -p, který zobrazuje pole lchwaits.

onstat -s

Provedením příkazu **onstat -s** získáte obecné informace o zámcích. Výstup obsahuje sloupec **userthread**, ve kterém, jsou zobrazeny adresy uživatelských jednotkových procesů čekajících na zámek. Tyto adresy můžete porovnat s uživatelskými adresami výstupu příkazu **onstat -u** a získat identifikační číslo uživatelského procesu.

Obrázek 4-13 zobrazuje vzorový výstup příkazu **onstat -s** .

```
...
Latches with lock or userthread set
name     address  lock wait userthread
LRU1     402e90   0    0        6b29d8
bf[34]   4467c0   0    0        6b29d8
...
```

Obrázek 4-13. Výstup příkazu onstat -s.

Monitorování zámků programem ISA

Chcete-li monitorovat zámky a zámky spin programem ISA, přejděte na stránku **Performance > Locks** a klepněte na položky **Latches** nebo **Spin Locks**. Program ISA používá k zobrazení informací informace generované příkazy **onstat -s** a **onstat -g spi**. Klepnutím na tlačítko **Refresh** spustíte příkazy znovu a zobrazíte aktuální informace.

Monitorování zámků pomocí tabulek SMI

Z tabulky **sysprofile** SMI získáte počet, kolikrát musel jednotkový proces čekat na zámek. Tato hodnota je uvedena ve sloupci **latchwts**

Šifrované hodnoty

Šifrované hodnoty zabírají v paměti více místa než odpovídající hodnoty v prostém textu, protože je s hodnotou nutné uložit všechny informace potřebné k jejímu dešifrování, kromě šifrovacího klíče. Většina šifrovaných dat vyžaduje více paměťového prostoru než data nešifrovaná. Vynecháním pokynu použitého s heslem můžete snížit režii šifrování až o 50 bajtů. Pokud používáte šifrované hodnoty, musíte se ujistit, že máte pro tyto hodnoty dostatek místa.

Poznámka: Vkládání nulových bajtů do výsledků šifrování se nedoporučuje.

Další informace o šifrování naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*. Také si projděte informace o vypočítávání paměťových požadavků šifrovaných dat pomocí příkazů ENCRYPT_TDES() aENCRYPT_AES() v *Příručce jazyka SQL: Syntax*.

Kapitola 5. Vliv konfigurace na aktivitu vstupu - výstupu

Obsah kapitoly

Konfigurace databázového serveru ovlivňuje aktivitu vstupu - výstupu několika způsoby. Přiřazení bloků a prostorů dbspace může vytvořit *aktivní body* vstupu - výstupu nebo oddíly disku s nepoměrně velikou aktivitou vstupu - výstupu. Přidělení kritických dat, řadicích oblastí a oblastí pro dočasné soubory a sestavení indexů může umístit intermittent loads na různé disky. Způsob konfigurace dopředného čtení může zvýšit efektivitu individuálních operací vstupu - výstupu. Způsob konfigurace úloh vstupu - výstupu na pozadí, jako například protokolování a vyčištění stránky, může ovlivnit propustnost vstupu - výstupu. O těchto tématech pojednávají následující části.

Konfigurace bloku a prostoru dbspace

Všechna data v databázi Informix jsou uložena na disku. Pro přístup k textovým nebo binárním datům typu BYTE a TEXT načteným z optického média Optical Subsystem používá také magnetický disk. Výkon aplikace určuje rychlost, s jakou je databázový server schopen kopírovat na disk nebo z disku vhodné datové stránky.

Disky jsou obvykle tou nejpomalejší komponentou cesty vstupu - výstupu při provedení transakce nebo dotazu na jednom hostitelském počítači. Síťová komunikace může také způsobit zpoždění v aplikacích typu klient/server. Tato zpoždění však administrátor databázového serveru obvykle nemůže ovlivnit. Informace o opatřeních, která administrátor databázového serveru může podniknout proto, aby zlepšil síťovou komunikaci, naleznete v částech “Společné oblasti vyrovnávacích pamětí v síti” na stránce 3-14 a “Připojení a využití CPU” na stránce 3-22.

Při častých dotazech uživatelů na stránky se může disk stát nadměrně používaným nebo se může přetížit. Disk se může přetížit v následujících případech:

- Disk je používán k několika účelům, například pro protokolování a aktivní databázové tabulky.
- Na disku jsou uložena různorodá data.

- Dojde k prokládání oblastí tabulek.

Optimální rozvržení disku, bloků a prostorů dbspace určí různé funkce, které vyžaduje aplikace spolu s funkcemi kontroly konzistence, které provádí databázový server. Čím více disků bude mít databázový server k dispozici, tím jednodušší mezi nimi bude vyvážit operace vstupu - výstupu. Další informace o těchto faktorech naleznete v části Kapitola 6, "Úvahy o výkonu tabulek", na stránce 6-1.

Tato část se zabývá důležitými problémy, které se týkají počáteční konfigurace bloků, prostorů dbspace a blobspace. Při plánování rozvržení bloků a prostorů dbspace na disku zvažte následující problémy:

- umístění a zrcadlení kritických dat
- rozložení zatížení
- omezení sporů
- snadnost zálohování a obnovy

Spolu s fragmentací pomocí kruhového dotazování lze bloky rozložit mezi disky a řadiče a ušetřit tak čas a zpracování chyb. Umístění více bloků na jeden disk může zvýšit propustnost.

Přímý vstup - výstup pro předpřipravené soubory pro bloky prostoru dbspace (pouze systém UNIX)

Při přidělování prostoru na disku můžete použít:

- *Předpřipravené* soubory uložené ve vyrovnávací paměti operačního systému
- Předpřipravené soubory, které používají přímý vstup - výstup a obcházejí vyrovnávací paměť souborového systému
- Přístup k disku bez vyrovnávací paměti, který se také nazývá diskový prostor *s přímým přístupem* (disk s přímým přístupem přenáší data přímo mezi pamětí databázového serveru a diskem bez kopírování dat).

I když je v systémech UNIX doporučeno používat k dosažení vyššího výkonu zařízení s přímým přístupem, můžete v systémech UNIX zlepšit výkon předpřipravených souborů, pokud povolíte přímý vstup - výstup pomocí konfiguračního parametru DIRECT_IO. (Chcete-li zjistit nejlepší výkon zařízení, proveďte srovnávací test pro systém s oběma typy zařízení pro rozvržení prostoru dbspace a tabulky.)

Přímý vstup - výstup obchází použití vyrovnávací paměti souborového systému a z toho důvodu je efektivnější při čtení a zápisu na disk (ve srovnání s čtením a zápisem, který pouze přistupuje k vyrovnávací paměti souborového systému). Při přímém vstupu - výstupu je obvykle třeba zarovnávat data k hranicím diskových sektorů. Přímý vstup - výstup také obvykle umožňuje použití asynchronního vstupu - výstupu jádra (KAIO - kernel asynchronous I/O), který umožňuje další zlepšení výkonu. Při použití přímého vstupu - výstupu a KAIO tam, kde je k dispozici, se může výkon předpřipravených souborů použitých pro bloky prostoru dbspace blížit výkonu zařízení s přímým přístupem.

Pokud používaný systém souborů podporuje přímý vstup - výstup pro velikost stránek, kterou používá blok prostoru dbspace, pracuje server Dynamic Server takto:

- Nepoužívá ve výchozím nastavení přímý vstup - výstup.
- Používá přímý vstup - výstup, pokud je konfigurační parametr DIRECT_IO nastaven na hodnotu 1.
- Používá ve výchozím nastavení KAIO (pokud jej souborový systém podporuje) s přímým vstupem - výstupem.

- Nepoužívá KAIO s přímým vstupem - výstupem, pokud je nastavena proměnná prostředí KAIOOFF.
- Nepoužívá přímý vstup - výstup pro dočasné prostory dbspace.

Přidružení diskových oddílů k blokům

Doporučuje se přiřadit bloky ke všem diskovým oddílům. Když se blok shoduje s diskovým oddílem (nebo se zařízením), je snadné sledovat velikost využití místa na disku a vyhnout se tak chybám z důvodu nesprávně vypočítaných posunutí. Maximální velikost bloku je 4 TB.

Přidružení prostorů dbspace k blokům

Doporučuje se přidružit jeden blok k prostoru dbspace, zejména tehdy, má-li být prostor dbspace použit pro fragment tabulky. Další informace o umístění a rozvržení tabulky naleznete v části Kapitola 6, "Úvahy o výkonu tabulek", na stránce 6-1.

Umístění tabulek systémového katalogu s tabulkami databáze

Když disk obsahující systémový katalog pro určitou databázi selže, celá databáze zůstane nepřístupná, dokud se systémový katalog neobnoví. Z důvodů této potenciální nedostupnost databáze se doporučuje, aby se všechny tabulky systémového katalogu shromažďovaly v jednom prostoru dbspace. Místo toho je vhodné umístit tabulky systémového katalogu spolu s tabulkami databáze, které popisují.

Vytvoření tabulky systémového katalogu v tabulce prostoru typu dbspace:

1. Vytvoření databáze v tabulce prostoru dbspace, kde má být tabulka uložena
2. Chcete-li z databáze udělat aktuální databázi, použijte příkazy SQL DATABASE nebo CONNECT.
3. Zadání příkazu CREATE TABLE pro vytvoření tabulky.

Umístění kritických dat

Disk nebo disky obsahující stránky rezervované systémem, fyzický protokol a prostory dbspace, které obsahují soubory logického protokolu, jsou pro operace databázového serveru kritické. Databázový server nemůže fungovat, pokud se některý z těchto prvků stane nedostupným. Databázový server ve výchozím nastavení umísťuje všechny tři kritické prvky v kořenovém prostoru dbspace.

Chcete-li dosáhnout vhodné umísťovací strategie pro kritická data, je nutné zvolit mezi dostupností dat a maximálním výkonem protokolování.

Databázový server v původním nastavení umisťuje do kořenového prostoru dbspace také dočasné tabulky a tabulky řazení. Doporučuje se použít konfigurační parametr DBSPACETEMP a proměnnou prostředí **DBSPACETEMP** a přiřadit těmto tabulkám a souborům jiné prostory dbspace. Podrobnosti naleznete v části "Konfigurace prostorů dbspace pro dočasné tabulky a soubory řazení" na stránce 5-7.

Zvážení možnosti použití oddělených disků pro komponenty s kritickými daty

Pokud umístíte kořenový prostor dbspace, logický protokol a fyzický protokol do rozdílných prostorů dbspace na oddělených discích, můžete dosáhnout některých výhod, které zřetelně zlepší výkon. Každý disk, který použijete pro komponentu s kritickými daty by měl mít svůj řadič. Tento přístup má následující výhody:

- Odděluje aktivitu protokolování od vstupu - výstupu databáze a umožňuje, aby byly paralelně prováděny požadavky vstupu - výstupu logického a fyzického protokolu.

- Snižuje dobu potřebnou pro obnovení při selhání

  Pokud však disky nejsou zrcadlené, existuje zvýšené nebezpečí, že disk, který obsahuje kritická data může být při selhání poškozen. To bude mít za následek zastavení databázového serveru a bude nutná kompletní obnova všech dat ze zálohy level-0.
- Pro relativně malý kořenový prostor dbspace, který obsahuje pouze rezervované stránky, oddíly databáze a databázi **sysmaster**.

  V mnoha případech postačuje 10 000 kB.

Databázový server používá ke konfiguraci rozdílných částí kritických dat různé metody. Chcete-li přiřadit kořenový prostoru dbspace a fyzický protokol vhodnému prostoru dbspace, nastavte konfigurační parametry vhodného databázového serveru. Chcete-li přiřadit soubory logického protokolu vhodnému prostoru dbspace, použijte obslužný program **onparams**.

Další informace o konfiguračních parametrech, které ovlivňují každou část kritických dat, naleznete v části “Konfigurační parametry, které ovlivňují kritická data” na stránce 5-7.

Zvážení možnosti použití zrcadlení pro komponenty kritických dat

Zvažte možnost použití zrcadlení pro prostory dbspace, které obsahují kritická data. Použití zrcadlení pro tyto prostory dbspace zajistí, že databázový server bude moct pokračovat dál ve své funkci, i když dojde k selhání jednoho disku. V závislosti na směsi požadavků vstupu - výstupu pro daný prostor dbspace existuje možnost volby mezi odolností vůči selhání zrcadlení a výkonem vstupu - výstupu. Zrcadlením prostorů dbspace, které jsou velmi často čtené, dosáhnete zřetelného zlepšení výkonu. Drobné snížení výkonu způsobí zrcadlení prostorů dbspace s častým zápisem.

Mnoho moderních paměťových zařízení disponuje vynikajícími možnostmi pro zrcadlení a lze je k zrcadlení využít namísto dynamického serveru.

Díky zrcadlení jsou pro provedení čtecích požadavků k dispozici dva disky a databázový server tak může těchto požadavků zpracovat více. Každý zapisovací požadavek ale vyžaduje k uskutečnění zápisu dvě operace a nedokončí se dříve, než jsou obě tyto operace provedeny. Operace zápisu se provádějí současně, požadavek však není dokončen dříve, než pomalejší z obou disků provede aktualizaci. Při zrcadlení prostorů dbspace s častým zápisem se proto projeví drobné snížení výkonu.

Zrcadlení kořenového prostoru dbspace

Pokud provedete zrcadlení kořenového prostoru dbspace a omezíte jeho obsah pouze na tabulky, ze kterých se jen čte, nebo se k ním přistupuje zřídka, dosáhnete při minimálním snížení výkonu jisté odolnosti vůči selhání. Když umístíte často aktualizované tabulky do jiných, nezrcadlených prostorů dbspace, můžete pro teplé obnovení těchto tabulek v případě selhání disku použít zařízení databázového serveru pro zálohování a obnovu. Když je kořenový prostor dbspace zrcadlený, databázový server zůstane po dobu opravy selhaného disku online a může provádět další transakce.

Když budete provádět zrcadlení kořenového prostoru dbspace, první blok vždycky umístěte na jiný disk než na ten, který slouží jako zrcadlo. Konfigurační parametr MIRRORPATH by měl mít jinou hodnotu než parametr ROOTPATH.

Zrcadlení bloků s inteligentními velkými objekty

Prostor typu sbspace je logická paměťová jednotka složená z jednoho nebo více bloků, do kterých jsou ukládány *inteligentní velké objekty*. Mezi inteligentní velké objekty patří objekty CLOB (znakové velké objekty) a objekty BLOB (binární velké objekty). Podrobnější popis prostorů typu sbspace naleznete v příručce *IBM Informix Administrator's Guide*.

První blok prostoru typu sbspace obsahuje speciální sadu stránek nazývaných *metadata*. Metadata slouží k vyhledávání inteligentních velkých objektů v prostoru typu sbspace. Další bloky, které jsou do prostoru typu sbspace přidány, mohou také mít stránky s metadaty. Musí však být určeny v příkazu **onspaces** při vytváření bloku.

Zvažte, zda zrcadlit bloky obsahující stránky s metadaty z následujících důvodů:

- Lepší dostupnost

  Bez přístupu ke stránkám s metadaty nemají uživatelé přístup k žádnému inteligentnímu velkému objektu v prostoru typu sbspace. Pokud se stane nedostupným disk, na kterém se nalézá první blok prostoru typu sbspace obsahující všechny stránky s metadaty, nelze v prostoru typu sbspace přistoupit k inteligentnímu velkému objektu, i když je uložen v bloku na jiném disku. Chcete-li dosáhnout lepší dostupnosti, zrcadlete alespoň první blok prostoru typu sbspace a jakýkoliv další blok obsahující stránky s metadaty.
- Rychlejší přístup

  Zrcadlením bloku obsahujícím stránky s metadaty lze rozšířit aktivitu čtení i na další disky obsahující primární a zrcadlený blok.

Zrcadlení logického protokolu

K zápisování logického protokolu dochází velmi často. Pokud prostor dbspace obsahuje soubory logického protokolu a je zrcadlený, dojde k nepatrnému zhoršení výkonu v důsledku dvojitého zapisování, které je popsáno v části “Zvážení možnosti použití zrcadlení pro komponenty kritických dat” na stránce 5-5. Výběrem vhodné velikosti vyrovnávací paměti a režimu protokolování však lze na určitou úroveň upravit rychlost, s jakou protokolování vytváří požadavky vstupu - výstupu.

Při protokolování bez vyrovnávací paměti a při protokolování kompatibilním se standardem ANSI požaduje po disku databázový server vyprázdnění vyrovnávací paměti protokolů u každé potvrzené transakce (dvakrát, pokud je prostor dbspace zrcadlený). Protokolování s vyrovnávací pamětí generuje daleko méně požadavků vstupu - výstupu než protokolování bez vyrovnávací paměti nebo protokolování kompatibilní se standardem ANSI.

U protokolování s vyrovnávací pamětí je vyrovnávací paměť protokolu zapsána na disk pouze tehdy, když je zaplněná a všechny transakce, které obsahuje jsou dokončené. Frekvenci vstupu - výstupu logického protokolu lze snížit ještě více, pokud se zvýší velikost vyrovnávací paměti logického protokolu. U protokolování s vyrovnávací pamětí jsou však v případě selhání systému transakce v jakékoliv částečně zaplněné vyrovnávací paměti náchylné ke ztrátě.

Přesto že je u protokolování s vyrovnávací pamětí zaručena konzistence databáze, určité transakce před selhání zabezpečeny nejsou. Čím větší je vyrovnávací paměť logického protokolu, tím více transakcí bude při obnově služby po selhání pravděpodobně potřeba znovu zadat.

Na rozdíl od fyzického protokolu nelze pro soubory logického protokolu v počáteční konfiguraci databázového serveru určit alternativní prostor dbspace. Místo toho použijte obslužný program **onparams** a nejprve přidejte soubory logického protokolu k alternativnímu prostoru dbspace. Potom z kořenového prostoru dbspace soubory logického protokolu vypusťte. Další informace o obslužném programu **onparams** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Zrcadlení fyzického protokolu

Při vyprazdňování dat z vyrovnávací paměti a při aktivitě spojené s kontrolními body dochází u fyzického protokolu k častému zápisu. Když je aktivován jednotkový proces vyčištění stránky, vyskytne se také vstup - výstup kdo fyzického protokolu. Pokud prostor dbspace obsahuje soubory fyzického protokolu a je zrcadlený, dojde k nepatrnému zhoršení

výkonu v důsledku dvojitého zapisování, které je popsáno v části "Zvážení možnosti použití zrcadlení pro komponenty kritických dat" na stránce 5-5.

Chcete-li udržet minimální vstup - výstup do fyzického protokolu, můžete upravit interval kontrolního bodu a minimální a maximální velikost prahu LRU. (Další informace naleznete v částech "CKPTINTVL" na stránce 5-28 a "BUFFERPOOL" na stránce 5-36.)

Konfigurační parametry, které ovlivňují kritická data

Chcete-li konfigurovat kořenový prostor dbspace, můžete použít následující parametry:

- ROOTNAME
- ROOTOFFSET
- ROOTPATH
- ROOTSIZE
- MIRROR
- MIRRORPATH
- MIRROROFFSET

Tyto parametry určují umístění a velikost počátečního bloku kořenového prostoru dbspace a konfigurují zrcadlení, pokud pro daný blok existuje. (Pokud je počáteční blok zrcadlený, všechny ostatní bloky v kořenovém prostoru dbspace musí být také zrcadlené). Jinak tyto parametry nebudou mít žádný významný vliv na výkon.

Logické protokoly ovlivňují následující konfigurační parametry:

- LOGSIZE
- LOGBUFF

Parametr LOGSIZE určuje velikost každého souboru logického protokolu. Parametr LOGBUFF určuje velikost tří vyrovnávacích pamětí logického protokolu ve sdílené paměti. Další informace o parametru LOGBUFF naleznete v části "LOGBUFF" na stránce 4-14.

Umístění a velikost fyzického protokolu určují následující konfigurační parametry:

- PHYSDBS
- PHYSFILE

Konfigurace prostorů dbspace pro dočasné tabulky a soubory řazení

Aplikace používající dočasné tabulky nebo velké řadicí operace vyžadují velké množství dočasného prostoru. Chcete-li zlepšit výkon těchto aplikací, použijte konfigurační parametr DBSPACETEMP nebo proměnnou prostředí **DBSPACETEMP** a označte jeden nebo více prostorů dbspace pro dočasné tabulky a soubory řazení.

V závislosti na způsobu vytvoření dočasného místa používá databázový server následující výchozí umístění pro dočasné tabulky a soubory řazení, není-li zadán parametr DBSPACETEMP:

- Prostor dbspace aktuální databáze, když vytvoříte explicitní dočasnou tabulku s klauzulí TEMP TABLE příkazu CREATE TABLE a neurčíte prostor dbspace pro tabulku ani v klauzuli IN prostoru dbspace, ani v klauzuli FRAGMENT BY

  Tato akce může vážně ovlivnit vstup - výstup do daného prostoru dbspace. Pokud je kořenový prostor dbspace zrcadlený, setkáte se s drobným snížení výkonu v důsledku dvojího zapisování vstupu - výstupu do dočasných tabulek a řadicích souborů.

- Kořenový prostor dbspace, když vytvoříte explicitní dočasnou tabulku pomocí volby INTO TEMP příkazu SELECT

  Tato akce může vážně ovlivnit vstup - výstup do kořenového prostoru dbspace. Pokud je kořenový prostor dbspace zrcadlený, setkáte se s drobným snížení výkonu v důsledku dvojího zapisování vstupu - výstupu do dočasných tabulek a řadicích souborů.
- Adresář nebo soubor operačního systému, který se určuje jednou z následujících proměnných:
  - V systému UNIX to je adresář nebo adresáře operačního systému, které proměnná prostředí **PSORT_DBTEMP**určuje, pokud je nastavená.

    Pokud proměnná prostředí **PSORT_DBTEMP** není nastavená, databázový server zapisuje soubory řazení do místa určeného pro soubory operačního systému v adresáři **/tmp** .
  - V systému Windows je to adresář určený v položce **TEMP** nebo **TMP** v okně uživatelské proměnné prostředí v nabídce **Ovládací panely > System.**

  Databázový sever používá adresář nebo soubory operačního systému k nasměrování jakéhokoliv přetečení způsobeného následujícími činnostmi databáze:
  - Příkaz SELECT s klauzulí GROUP BY
  - Příkaz SELECT s klauzulí ORDER BY
  - Operace Hash-join
  - Operace Nested-loop join
  - Vytváření indexů

Upozornění: Pokud není určena hodnota pro konfigurační parametr DBSPACETEMP nebo pro proměnnou prostředí **DBSPACETEMP**, databázový server tyto soubory operačního systému použije pro implicitní dočasné tabulky. Pokud je v tomto systému souborů nedostatek místa k uložení řadicího souboru, dotaz provádějící řazení vrátí chybovou zprávu. Operační systém by mohl být vážně zasažený, dokud nebude řadicí soubor odstraněn.

Výkon se dá zlepšit tím, že se vytvoří dočasné prostory dbspace výhradně pro ukládání dočasných tabulek a řadicích souborů. Doporučuje se použít konfigurační parametr DBSPACETEMP a proměnnou prostředí **DBSPACETEMP** a přiřadit těmto tabulkám a souborům prostory dbspace.

Když se prostor dbspace určí buď v konfiguračním parametru DBSPACETEMP nebo v proměnné prostředí **DBSPACETEMP** , získají se následující výhody ve výkonnosti:

- Sníží se dopad vstupu - výstupu na kořenový prostor dbspace, na tvorbu prostorů dbspace nebo na soubory operačního systému.
- Využije se paralelního řazení do dočasných souborů (pro zpracování dotazovacích klauzulí, jako jsou například ORDER BY nebo GROUP BY, nebo pro řazení indexových klíčů při provádění parametru CREATE INDEX), když je určen více než jeden prostor dbspace pro dočasné tabulky a hodnota priority PDQ je větší než 0.
- Zvýší se rychlost s jakou databázový server vytváří dočasné tabulky, když se dva nebo více prostorů dbspace přiřadí na samostatné disky.
- Využije se paralelního vkládání do dočasné tabulky, když je hodnota priority PDQ větší než 0 a dočasná tabulka je vytvořena podle jedné z následujích specifikací uvedených v následující tabulce.

  Databázový server automaticky použije svých schopností pro paralelní vkládání a rozdělí dočasnou tabulku do těchto prostorů dbspace pomocí schématu rozdělení kruhovým dotazováním.

Příkaz, který vytvoří dočasnou tabulka	Databáze je přihlášená	Klauzule WITH NO LOG	Rozdělení pomocí klauzule	Místo vytvoření dočasné tabulky
CREATE TEMP TABLE	Ano	Ne	Ne	Kořenový prostor dbspace
CREATE TEMP TABLE	Ano	Ano	Ne	Jeden z prostorů dbspace určený v proměnné prostředí DBSPACETEMP
CREATE TEMP TABLE	Ano	Ne	Ano	Nelze vytvořit dočasnou tabulku. Chyba 229/196

Důležité: Chcete-li dosáhnout lepšího výkonu operací řazení a zabránit databázovému serveru v neočekávaném vyplňování systémů souborů, doporučuje se použít parametr DBSPACETEMP nebo proměnnou prostředí **DBSPACETEMP**. Vypsané prostory dbspace se musí skládat z bloků, které jsou přiděleny jako zařízení bez vyrovnávací paměti.

Vytvoření dočasných prostorů typu dbspace

Chcete-li vytvořit prostor dbspace pro výlučné použití dočasných tabulek a řadicích souborů, použijte volbu **onspaces -t**. Chcete-li dosáhnout nejlepšího výkonu, postupujte podle následujících pravidel:

- Pokud vytváříte více než jeden dočasný prostor dbspace, vytvořte každý tento prostor na samostatném disku, abyste vyvážili vliv vstupu - výstupu.
- Na jeden disk umístěte vždy jen jeden dočasný prostor dbspace.

Databázový server neprovádí logické ani fyzické protokolování dočasných prostorů dbspace a dočasné prostory dbspace nejsou nikdy zálohovány jako součást celosystémové zálohy. Dočasný prostor dbspace vytvořený pomocí **onspaces -t** nelze zrcadlit.

Důležité: V případě databáze s protokolováním je nutné do příkazu SELECT... INTO TEMP zahrnout také klauzuli WITH NO LOG, aby se explicitní dočasné tabulky umístily do prostorů dbspace uvedených v konfiguračním parametru DBSPACETEMP a v proměnné prostředí **DBSPACETEMP**. Jinak databázový server uloží explicitní dočasné tabulky do kořenového prostoru dbspace.

Další informace týkající se vytvoření dočasných prostorů dbspace naleznete v příručce *IBM Informix Administrator's Guide*.

Konfigurační parametr DBSPACETEMP

Konfigurační parametr DBSPACETEMP určuje seznam prostorů dbspace, do kterých databázový server ve výchozím nastavení umisťuje dočasné tabulky a soubory řazení. Některé nebo všechny prostory dbspace, které v tomto konfiguračním parametru vypíšete, mohou být dočasnými prostory dbspace, které jsou rezervované výhradně k uložení dočasných tabulek a řadicích souborů.

Pokud v tomto seznamu určíte více než jeden prostor dbspace, databázový server použije svých schopností pro paralelní vkládání a rozdělí dočasné tabulky do všech vypsaných prostorů dbspace pomocí schématu rozdělení kruhovým dotazováním. Další informace naleznete v části “Navržení schématu distribuce” na stránce 9-6.

Konfigurační parametr DBSPACETEMP dovoluje administrátorovi databáze omezit, které prostory dbspace databázový server použije pro dočasné úložiště. Podrobné informace o nastavení parametru DBSPACETEMP naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Důležité: Konfigurační parametr DBSPACETEMP není nastaven v souboru **onconfig.std**. Chcete-li dosáhnout nejlepšího výkonu s dočasnými tabulkami a řadicími soubory, použijte parametr DBSPACETEMP a určete dva nebo více prostorů dbspace na samostatných discích.

Proměnná prostředí DBSPACETEMP

Chcete-li přepsat parametr DBSPACETEMP, můžete použít proměnnou prostředí **DBSPACETEMP** pro dočasné tabulky a soubory řazení. Tato proměnná prostředí určuje seznam prostorů dbspace oddělený čárkou nebo dvojtečkou, do kterých se umístí dočasné tabulky pro aktuální relaci.

Důležité: Chcete-li dosáhnout lepšího výkonu operací řazení a zabránit databázovému serveru v neočekávaném vyplňování systémů souborů, doporučuje se použít parametr DBSPACETEMP nebo proměnnou prostředí **DBSPACETEMP**.

Chcete-li určit soubory řazení, použijte z následující důvodů raději parametr DBSPACETEMP než proměnnou prostředí **PSORT_DBTEMP**:

- Parametr **DBSPACETEMP** přináší obvykle lepší výkon.

  Pokud jsou prostory dbspace umístěny na znakových speciálních zařízeních (známé také jako disková zařízení s přímým přístupem), databázový server použije přístup k disku bez vyrovnávací paměti. Operace vstup - výstup je rychlejší u zařízení bez vyrovnávací paměti než u běžných souborů operačního systému (s vyrovnávací pamětí), protože databázový server spravuje operace vstupu - výstupu okamžitě.
- Proměnná prostředí **PSORT_DBTEMP** určuje jeden nebo více adresářů operačního systému, kam se umisťují soubory řazení.

  Tyto soubory operačního systému se mohou v počítači neočekávaně vyplnit, protože je databázový server nespravuje.

Odhad dočasného prostoru

Chcete-li odhadnout, jaké množství dočasného prostoru přidělit, použijte následují pravidla:

- Aplikacím OLTP přidělte dočasné prostory dbspace, které se rovnají minimálně 10 procentům tabulky.
- Aplikacím DSS přidělte dočasné prostory dbspace, které se rovnají minimálně 50 procentům tabulky.

Spojení typu hash join, které pracuje tak, že vytvoří tabulku (hashovací tabulku) z řádků jedné z tabulek spojení a prohledává ji s řádky druhé tabulky, může využívat značné množství paměti a potenciálně může způsobit přetečení do dočasného prostoru na disku. Velikost hashovací tabulky je určena velikostí tabulky použité k jejímu vytvoření (hashovací tabulka je často menší, než dvě tabulky spojení), po použití všech filtrů, které mohou omezit počet řádků a případně snížit počet sloupců.

Chcete-li odhadnout, jaké množství paměti vyžaduje hashovací tabulka ve spojení hash join, použijte následují vzorec:

```
velikost_hashovací_tabulky = (32 bajtů + velikost_řádků_malé_tabulky) *
     počet_řádků_malé_tabulky
```

kde hodnoty velikost_řádků_malé_tabulky a počet_řádků_malé_tabulky označují velikost a počet řádků v menší z obou tabulek, které se spojení účastní.

Informace o spojeních typu hash join naleznete v části “Spojení typu hash” na stránce 10-4.

Chcete-li konfigurovat paměť řazení pro všechny dotazy kromě dotazů PDQ, můžete použít konfigurační parametr DS_NONPDQ_QUERY_MEM. Další informace naleznete v příručce “Více paměti pro dotazy se hashovacími spojeními, souhrny a dalšími prvky náročnými na paměť” na stránce 13-21

Proměnná prostředí PSORT_NPROCS

Proměnná prostředí**PSORT_NPROCS** určuje maximální počet podprocesů, které může databázový server použít k seřazení dotazu. Pokud dotaz vyžaduje velkou operaci řazení, můžou se paralelně provést vícenásobné řadicí podprocesy, aby se zlepšil výkon dotazu.

Pokud je hodnota priority PDQ rovna 0 a **PSORT_NPROCS** je větší než 1, databázový server použije paralelní řazení. Správa PDQ tato řazení neomezuje. Jinými slovy, ačkoliv je řazení prováděno paralelně, databázový server řazení nepovažuje jako aktivitu PDQ. Pokud je priorita PDQ 0, databázový server neřídí řazení žádným z konfiguračních parametrů PDQ.

Pokud je priorita PDQ větší než 0 a proměnná prostředí **PSORT_NPROCS** je větší než 1, dotaz využije paralelní řazení i vlastnosti PDQ, jako je paralelní prohledávání a přídavná paměť. Uživatelé můžou použít proměnnou prostředí **PDQPRIORITY** k tomu, aby požádali o určitou část zdrojů PDQ pro dotaz. Pomocí parametru MAX_PDQPRIORITY můžete počet takovýchto požadavků uživatelů omezit. Další informace o parametru MAX_PDQPRIORITY naleznete v části “Omezení zdrojů PDQ v dotazech pomocí konfiguračního parametru MAX_PDQPRIORITY” na stránce 3-10.

Databázový server přiděluje pro řazení relativně malé množství paměti a tato paměť je rozdělena podle řadicích podprocesů **PSORT_NPROCS**. Řadicí procesy používají v případě nedostatku přidělené paměti dočasný prostor na disku. Další informace o paměti přidělené pro řazení naleznete v části “Odhad paměti potřebné pro řazení” na stránce 7-13.

Důležité: Chcete-li dosáhnout lepšího výkonu operace řazení, nastavte zpočátku hodnotu PSORT_NPROCS na 2, pokud má váš počítač několik CPU. Pokud je následná aktivita CPU nižší než aktivita vstupu - výstupu, můžete hodnotu PSORT_NPROCS zvýšit.

Další informace o řazení během vytváření rejstříků naleznete v části “Zvýšení výkonu při vytváření indexů” na stránce 7-12.

Konfigurace prostorů Sbspace pro dočasné inteligentní velké objekty

Aplikace mohou používat dočasné inteligentní velké objekty pro text, obrázky nebo pro další uživatelem definované datové typy pouze tehdy, pokud jsou vyžadovány během trvání uživatelovy relace. Tyto aplikace nevyžadují protokolování dočasných inteligentních velkých objektů. Protokolování přidá vstupní - výstupní aktivitu do logického protokolu a zvýší využití paměti.

Dočasné inteligentní velké objekty můžete ukládat do trvalého prostoru sbspace nebo do dočasného prostoru sbspace.

- Trvalé prostory sbspace

  Pokud ukládáte inteligentní velké objekty do běžného prostoru sbspace a zachováte výchozí atribut bez protokolování, změny objektů protokolovány nebudou, ale metadata budou vždy protokolována.

- Dočasné prostory sbspace

  Aplikace, které aktualizují dočasné inteligentní velké objekty uložené v dočasných prostorech typu sbspace jsou podstatně rychlejší, protože databázový server neprotokoluje metadata nebo uživatelská data v dočasném prostoru typu sbspace.

Chcete-li zlepšit výkon aplikací, které aktualizují dočasné inteligentní velké objekty, určete příznak **LOTEMP** ve funkci **mi_lo_specset_flags nebo ifx_lo_specset_flags** rozhraní API a určete dočasný prostor typu sbspace pro dočasné inteligentní velké objekty. Databázový server používá pro výběr umístění dočasných inteligentních velkých objektů následující pořadí podle priority:

- Prostor typu sbspace, který určíte ve funkci **mi_lo_specset_sbspace** nebo **ifx_lo_specset_sbspace** rozhraní API, když vytvoříte velký inteligentní objekt.

  Určete dočasný prostor typu sbspace ve funkci rozhraní API tak, aby se změny objektů a metadat neprotokolovaly. Prostor typu sbspace, který určíte ve funkci rozhraní API, přepíše každý výchozí prostor typu sbspace, který by mohl konfigurační parametr SBSPACETEMP nebo SBSPACENAME určovat.
- Prostor typu sbspace, který určíte v klauzuli IN prostoru Sbspace, když vytvoříte explicitní dočasnou tabulku s klauzulí TEMP TABLE příkazu CREATE TABLE

  Určete dočasný prostor typu sbspace v klauzuli IN prostoru Sbspace tak, aby se změny objektů a metadat neprotokolovaly.
- Trvalý prostor typu sbspace, který určíte v konfiguračním parametru SBSPACENAME, pokud neurčíte prostor typu sbspace v konfiguračním parametru SBSPACETEMP.

Pokud není ani v jedné z výše uvedených metod určený dočasný prostor typu sbspace, databázový server při pokusu o vytvoření dočasného inteligentního velkého objektu vydá následující chybovou zprávu:

```
-12053  Inteligentní velké objekty: Není určen počet prostorů sbspace.
```

Vytvoření dočasných prostorů Sbspace

Chcete-li vytvořit prostor typu sbspace k výlučnému použití pro dočasné inteligentní velké objekty, použijte obslužný program **onspaces -c -S** s volbou **-t**. Chcete-li dosáhnout nejlepšího výkonu, postupujte podle následujících pravidel:

- Pokud vytváříte více než jeden dočasný prostor typu sbspace, vytvořte každý tento prostor na samostatném disku, abyste vyvážili vliv vstupu - výstupu.
- Na jeden disk umístěte vždy jen jeden dočasný prostor typu sbspace.

Databázový server neprovádí logické ani fyzické protokolování dočasných prostorů typu sbspace a dočasné prostory typu sbspace nejsou nikdy zálohovány jako součást celosystémové zálohy. Dočasný prostor typu sbspace vytvořený pomocí **onspaces -t** nelze zrcadlit.

Důležité: V případě databáze s protokolováním je nutné do příkazu SELECT... INTO TEMP zahrnout klauzuli WITH NO LOG, aby se dočasné inteligentní velké objekty umístily do prostorů typu sbspace uvedených v konfiguračním parametru SBSPACETEMP. Jinak databázový server uloží dočasné inteligentní velké objekty do prostoru typu sbspace uvedeném v konfiguračním parametru SBSPACENAME.

Další informace o tom, jak vytvořit dočasné prostory typu sbspace naleznete v příručce *IBM Informix Administrator's Guide*. Další informace o tom, jak vytvořit dočasné inteligentní velké objekty naleznete v příručce *IBM Informix DataBlade API Programmer's Guide*.

Konfigurační parametr SBSPACETEMP

Konfigurační parametr SBSPACETEMP určuje seznam prostorů typu sbspace, do kterých databázový server ve výchozím nastavení umisťuje dočasné inteligentní velké objekty. Některé nebo všechny prostory typu sbspace, které v tomto konfiguračním parametru vypíšete, mohou být dočasnými prostory typu sbspace, které jsou rezervované výhradně k uložení dočasných inteligentních velkých objektů.

Konfigurační parametr SBSPACETEMP dovoluje administrátorovi databáze omezit, které prostory typu sbspace databázový server použije pro dočasné úložiště. Podrobné informace o nastavení parametru SBSPACETEMP naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Důležité: Konfigurační parametr SBSPACETEMP není nastaven v souboru **onconfig.std** file. Chcete-li dosáhnout nejlepšího výkonu s dočasnými inteligentními velkými objekty, použijte parametr SBSPACETEMP a určete dva nebo více prostorů typu sbspace na samostatných discích.

Umístění jednoduchých velkých objektů

Jednoduché velké objekty můžete uložit buď ve stejném prostoru dbspace, ve kterém se nalézá tabulka, nebo v prostoru blobspace.

Prostor blobspace je logická paměťová jednotka obsahující jeden nebo více bloků, do kterých jsou ukládány pouze jednoduché velké objekty (data typu TEXT a BYTE). Další informace o prostorech sbspace, které ukládají inteligentní velké objekty (jako například BLOB, CLOB a data s vícenásobnou reprezentací), najdete v části “Parametry, které ovlivňují vstup - výstup pro inteligentní velké objekty” na stránce 5-17.

Pokud použijete prostor blobspace, můžete jednoduché velké objekty uložit na jiný disk, než je uložena tabulka, ke které jsou data přidružená. Jednoduché velké objekty přidružené rozdílným tabulkám můžete uložit do stejného prostoru blobspace.

Prostor blobspace můžete vytvořit pomocí následujících obslužných programů:

- ON–Monitor (pouze systém UNIX)
- **ISA**
- **onspaces**

Podrobné informace o těchto obslužných programech naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Jednoduché velké objekty přiřadíte prostoru blobspace, když vytvoříte tabulky, ke kterým jsou jednoduché velké objekty přidružené. Více informací o příkazu SQL CREATE TABLE naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Jednoduché velké objekty nejsou protokolované a nepředávají se skrz společnou oblast vyrovnávací paměti. Nicméně, frekvence kontrolních bodů může ovlivnit aplikace, které k datům typu TEXT a BYTE přistupují. Další informace naleznete v části “LOGSIZE a LOGFILES” na stránce 5-28.

Výhoda prostorů blobspace oproti prostorům Dbspace

Když uložíte jednoduché velké objekty do prostoru blobspace nacházejícím se na jiném disku, než je uložena tabulka, ke které je prostor blobspace přidružený, databázový server poskytne následující výkonnostní výhody:

- Přístup k tabulce a jednoduchým velkým objektům je paralelní.

- Na rozdíl od jednoduchých velkých objektů uložených v prostoru dbspace jsou data prostoru blobspace zapsána přímo na disk. Jednoduché velké objekty se nepředávají skrz rezidentní sdílenou paměť, což nechává stránky paměti volné pro další použití.
- Jednoduché velké objekty nejsou protokolované, což snižuje aktivitu protokolování vstupu - výstupu protokolovaných databází.

Další informace naleznete v části "Ukládání jednoduchých velkých objektů v odděleném prostoru blobspace" na stránce 6-9.

Úvahy o velikosti stránky Blobpage

Prostory blobspace se dělí na jednotky zvané stránky *blobpage*. Databázový server načte jednoduché velké objekty z prostoru blobspace v jednotkách přizpůsobených velikosti stránky blobpage. Velikost stránky blobpage se při vytvoření prostoru blobspace zadává v násobcích stránky disku. Optimální velikost stránky blobpage pro konfiguraci závisí na následujících faktorech:

- Rozdělení velikosti jednoduchých velkých objektů.
- Poměr mezi rychlostí načtení největšího jednoduchého velkého objektu a množstvím místa na disku, které je nevyužito kvůli uložení jednoduchých velkých objektů do velkých stránek blobpage.

Chcete-li načíst jednoduchý velký objekt co nejrychleji, použijte velikost největšího jednoduchého velkého objektu zaokrouhlenou na nejbližší přírustek přizpůsobený velikosti stránky. Toto schéma zaručí, že databázový server dovede načíst i ten největší jednoduchý velký objekt pomocí jediného požadavku na vstup - výstup. Přestože toto schéma zajistí nejrychlejší načtení, má sklon k nevyužití místa na disku (plýtvání). Protože jsou jednoduché velké objekty uložené ve vlastních stránkách blobpage (nebo v sadě stránek blobpage), databázový server vyhrazuje stejné množství místa na disku pro každou stránku blobpage, i když jednoduchý velký objekt zabírá jen zlomek velikosti této stránky. Použití menší stránky blobpage umožní lepší využití disku, zvláště pokud existují velké rozdíly mezi velikostmi jednoduchých velkých objektů.

Chcete-li dosáhnout teoreticky největšího využití místa na disku, stránky blobpage by měly mít stejnou velikost jako standardní stránka disku. Potom by mnoho, ne-li většina jednoduchých velkých objektů, vyžadovala několik stránek blobpage. Protože databázový server načte zámek a vydá oddělený dotaz vstup - výstup pro každou stránku blobpage, toto schéma se provádí jen zřídka.

Ve skutečnosti používá vyvážené schéma pro změnu velikosti jako velikost stránky blobpage velikost nejčastěji se vyskytujícího jednoduchého velkého objektu. Předpokládejme například, že máte 160 hodnot jednoduchých velkých objektů v tabulce s následujícím rozdělením velikosti:

- 120 z těchto hodnot má velikost po 12 kB.
- Každá z dalších 40 hodnot má velikost 16 kB.

Můžete vybrat jednu z následujících velikostí stránky blobpage:

- Velikost 12kB stránky blobpage poskytuje větší efektivitu úložiště než velikost 16kB stránky blobpage, jak dokazují následující výpočty:
  - 12 kB

    Tato konfigurace umožňuje, aby většina hodnot jednoduchých velkých objektů vyžadovala jednu stránku blobpage a aby ostatních 40 hodnot vyžadovalo dvě stránky blobpage. V této konfiguraci je pro každou z větších hodnot v druhé stránce blobpage nevyužitých 8 kilobajtů. Celkové nevyužité místo je následující:

    ```
    nevyužité místo = 8 kB * 40
                    = 329 kB
    ```

– 16 kB

  V této konfiguraci jsou nevyužité 4 kB v oblastech 120 jednoduchých velkých objektů. Celkové nevyužité místo je následující:

```
nevyužité místo = 4 kB * 120
                = 640 kB
```

- Pokud aplikace přistupují k hodnotám 16kB jednoduchých velkých objektů častěji, databázový server musí provést samostatnou operaci vstupu - výstupu pro každou stránku blobpage. V tomto případě poskytuje 16kB velikost stránky blobpage lepší rychlost načtení než 12kB velikost stránky blobpage.

Tip: Pokud tabulka obsahuje více *sloupců typu jednoduchý velký objekt a hodnoty dat v těchto sloupcích mají různou velikost, uložte* data do různých prostorů blobspace tak, aby každý měl vhodnou velikost stránky.

Optimalizace velikosti stránky blobpage prostoru blobspace

Když vyhodnocujete strategii úložiště prostoru blobspace, efektivitu můžete měřit podle dvou kritérií:

- Zaplnění stránky blobpage.
- Stránky blobpage vyžadované jednoduchým velkým objektem.

Zaplnění stránky blobpage odkazuje na množství dat v každé stránce blobpage. Data typu TEXT a BYTE uložené v prostoru blobspace nemůžou sdílet stránky blobpage. Pokud tedy vyžaduje jednoduchý velký objekt pouze 20 procent stránky blobpage, zbývajících 80 procent stránky zůstane nedostupných. Vyhnutí se tomu však způsobí, že stránky blobpage budou příliš malé. Když je k uložení každého jednoduchého velkého objektu potřeba několika stránek blobpage, zvýší se režijní náklady úložiště. Například, pro aktualizace je vyžadováno více zámků, protože jeden zámek se musí získat pro každou stránku blobpage.

Získání statistických údajů o paměti prostoru blobspace

Při určování optimální velikosti stránek blobpage všech prostorů blobspace může pomoci následující příkaz **oncheck -pB**. Příkaz **oncheck -pB** zobrazuje následující statistické údaje o každé tabulce (nebo databázi):

- Počet stránek blobpage používaných tabulkou (nebo databází) v každém z prostorů blobspace.
- Průměrné zaplnění stránek blobpage používaných všemi jednoduchými velkými objekty uloženými jako součást tabulky (nebo databáze).

Určení zaplnění stránky blobpage pomocí příkazu oncheck -pB

Příkaz **oncheck -pB** zobrazuje statistické údaje, popisující průměrné zaplnění stránek blobpage. Tyto statistické údaje poskytují ukazatel efektivity úložiště pro jednotlivé jednoduché velké objekty v databázi nebo v tabulce. Pokud zjistíte, že statistické údaje udávají pro značný počet jednoduchých velkých objektů nízké procento zaplnění, databázový server by pravděpodobně zaznamenal zvýšení výkonu při změně velikosti stránky blobpage prostoru blobspace.

Oba výstupy **oncheck -pB** a **onstat -d update** zobrazují stejné údaje o počtu volných stránek blobpage. Další informace o příkazu **onstat -d update** naleznete v části týkající se správy místa na disku v příručce *IBM Informix Administrator's Guide*.

Spusťte příkaz **oncheck -pB** a jako parametr zadejte název databáze nebo tabulky. V následujícím příkladu jsou získány informace o všech jednoduchých velkých objektech uložených v tabulce **sriram.catalog** databáze **stores_demo**:

```
oncheck -pB stores_demo:sriram.catalog
```

Obrázek 5-1 zobrazuje výstup tohoto příkazu.

```
             BLOBSpace Report for stores_demo:sriram.catalog

Total pages used by table           7

BLOBSpace usage:
Space   Page             Percent Full
Name    Number    Pages  0-25%  26-50%  51-75  76-100%
------------------------------------------------------------
blobPIC 0x300080  1      x
   blobPIC 0x300082  2      x
         ------
Page Size is 6144      3

bspc1   0x2000b2  2             x
bspc1   0x2000b6  2                     x
              ------
Page Size is 2048      4
```

Obrázek 5-1. Výstup příkazu oncheck -pB

Space Name je název prostoru blobspace, který obsahuje jeden nebo více jednoduchých velkých objektů uložených jako část tabulky (nebo databáze).

Page Number je počáteční adresa v prostoru blobspace specifického jednoduchého velkého objektu.

Pages je počet stránek databázového serveru potřebných k uložení tohoto jednoduchého velkého objektu.

Percent Full je ukazatel průměrného zaplnění stránky blobpage, podle prostoru blobspace, pro každý prostor blobspace v této tabulce nebo databázi.

Page Size je velikost stránky blobpage v bajtech pro tento prostor blobspace. Velikost stránky blobpage je vždy rovna násobku velikosti stránky databázového serveru.

Příklad výstupu ukazuje, že čtyři jednoduché velké objekty jsou uloženy jako část tabulky **sriram.catalog**. Dva objekty jsou uloženy v prostoru blobspace **blobPIC** ve stránkách blobpage o velikosti 6144 bajtů. Dva další objekty jsou uloženy v prostoru blobspace **bspc1** ve stránkách blobpage o velikosti 2048 bajtů.

Souhrný údaj, který se zobrazí v horní části zobrazení, **Total pages used by table** (Celkový počet stránek použitých tabulkou) je prostý součet stránek blobpage, kterých je potřeba k uložení jednoduchých velkých objektů. Součet ale neříká nic o velikosti použitých stránek blobpage, o počtu uložených jednoduchých velkých objektů, ani o celkovém počtu uložených bajtů.

Údaj o efektivitě zobrazený pod nadpisem **Percent Full** (Procento zaplnění) je nepřesný, může ale administrátora upozornit na trendy v ukládání dat typu TEXT a BYTE.

Interpretace průměrného zaplnění stránky blobpage: Tato část demonstruje myšlenku průměrného zaplnění. První jednoduchý velký objekt, který znázorňuje Obrázek 5-1 na stránce 5-16, je uložený v prostoru blobspace **blobPIC** a vyžaduje jednu stránku blobpage o velikosti 6144 bajtů. Stránka blobpage je zaplněna z 51 až 75 procent, což znamená, že její velikost je mezi 0,51 * 6144 = 3133 bajty a 0,75 * 6144 = 4608 bajty. Maximální velikost tohoto jednoduchého velkého objektu musí být menší nebo rovna 75 procentům z 6144 bajtů, nebo 4608 bajtů.

Druhý objekt vypsaný v prostoru blobspace **blobPIC** vyžaduje k uložení dvě stránky blobpage o velikosti 6144 bajtů, nebo celkově 12 288 bajtů. Průměrné zaplnění všech přidělených stránek blobpage je 51 až 75 procent. Proto musí být minimální velikost objektu větší než 50 procent z 12 288 bajtů, nebo 6144 bajtů. Maximální velikost jednoduchého velkého objektu musí být menší nebo rovna 75 procentům z 12 288 bajtů, nebo 9216 bajtů. Průměrné zaplnění neznamená, že je každá stránka zaplněna mezi 51 a 75 procenty. Výsledek výpočtu průměrného zaplnění dvou stránek blobpage by byl mezi 51 a 75 procenty, i když by první stránka blobpage byla zaplněna na 100 procent a druhá stránka blobpage jen mezi 2 až 50 procenty.

Uvažujme nyní dva jednoduché velké objekty v prostoru blobspace **bspc1**. Oba dva objekty se zdají přibližně stejně veliké. Oba vyžadují dvě stránky blobpage o velikosti 2048 bajtů a průměrné zaplnění pro každý z nich 76 až 100 procent. Minimální velikost těchto jednoduchých velkých objektů musí být větší než 75 procent přidělených stránek blobpage, nebo 3072 bajtů. Maximální velikost pro každý objekt je nepatrně menší než 4096 bajtů (umožnění režie).

Použití kritéria efektivity na výstup: Při pohledu na údaj o efektivitě prostoru blobspace **bspc1** by se administrátor databázového serveru mohl rozhodnout, že pro ukládání dat typu TEXT a BYTE bude vhodnější zdvojnásobit velikost stránky blobpage z 2048 bajtů na 4096 bajtů. (Velikost stránky blobpage je vždy rovna násobku velikosti stránky databázového serveru.) Pokud by administrátor databázového serveru tuto změnu provedl, ukazatel zaplnění stránky by zůstal stejný, počet zámků potřebných během aktualizace jednoduchého velkého objektu by se ale snížil na polovinu.

Údaj o efektivitě stránky blobpage **blobPIC** nedává žádný zřetelný návrh ke zlepšení. Dva jednoduché velké objekty v **blobPIC** se značně liší ve velikosti a není pro ně žádná optimální strategie ukládání. Obecně můžou být jednoduché velké objekty podobné velikosti uloženy mnohem efektivněji než jednoduché velké objekty rozdílné velikosti.

Parametry, které ovlivňují vstup - výstup pro inteligentní velké objekty

Prostor sbspace je logická paměťová jednotka obsahující jeden nebo více bloků , do kterých mohou být ukládány jednoduché velké objekty (jako například BLOB, CLOB a data s vícenásobnou reprezentací). Další informace o prostorech sbspace naleznete v kapitole týkající se ukládání dat v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

V této kapitole jsou uvedeny pokyny pro následující témata:

- Rozvržení disků
- Konfigurační parametry
- Volby pro obslužný program **onspaces**

Programovací rozhraní aplikací DataBlade API a ESQL/C poskytuje také funkce, které ovlivní vstupní - výstupní operace pro inteligentní velké objekty.

Důležité: Pro většinu aplikací se doporučuje použít hodnoty vypočtené databázovým serverem k získání informací o paměti disku. Další informace naleznete v příručce *IBM Informix ESQL/C Programmer's Manual* a v příručce *IBM Informix DataBlade API Programmer's Guide*.

Rozložení disku pro prostory sbspace

Prostory sbspace vytvořte na jiných discích, než je uložena tabulka, ke které jsou data přidružena. Inteligentní velké objekty přidružené rozdílným tabulkám můžete uložit do stejného prostoru sbspace.

Pokud uložíte inteligentní velké objekty do prostoru sbspace nacházejícím se na jiném disku, než je uložena tabulka, ke které je prostor sbspace přidružený, databázový server poskytne následující výkonnostní výhody:

- Přístup k tabulce a inteligentním velkým objektům je paralelní.
- Pokud se rozhodnete nezapsat data do prostoru sbspace, snížíte aktivitu protokolování vstupu - výstupu u protokolovaných databází.

Chcete-li vytvořit prostor sbspace, použijte obslužný program **onspaces**, jak popisuje příručka *IBM Informix Dynamic Server Administrator's Reference*. Inteligentní velké objekty přiřadíte prostoru typu sbspace, když vytvoříte tabulky, ke kterým jsou inteligentní velké objekty přidružené. Více informací o příkazu SQL CREATE TABLE naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Konfigurační parametry, které ovlivňují vstup - výstup prostoru sbspace

Výkon vstupu - výstupu prostorů sbspace ovlivňují následující konfigurační parametry:

- SBSPACENAME
- BUFFERPOOL
- LOGBUFF

Parametr SBSPACENAME

Konfigurační parametr SBSPACENAME určuje výchozí název prostoru sbspace, pokud nezadáte název prostoru sbspace při definování sloupce dat typu CLOB nebo BLOB. Chcete-li snížit soupeření disku a zajistit lepší vyrovnání zátěže, umístěte výchozí prostor sbspace na jiný disk, než na kterém jsou uložena data tabulky.

BUFFERPOOL

Velikost společné oblasti vyrovnávací paměti ovlivňuje operace vstupu - výstupu pro inteligentní velké objekty, protože společná oblast vyrovnávací paměti je pro tyto objekty výchozí oblastí sdílené paměti. Pokud aplikace často přistupuje k velkým inteligentním objektům, je výhodné mít tyto objekty ve společné oblasti vyrovnávací paměti. Inteligentní velké objekty používají pouze společnou oblast vyrovnávací paměti o výchozí velikosti stránky. Další informace o odhadu zvýšení velikosti společné oblasti vyrovnávací paměti pro inteligentní velké objekty naleznete v části “Inteligentní velké objekty a vyrovnávací paměť” na stránce 4-11.

Ve výchozím nastavení databázový server vkládá inteligentní velké objekty do vyrovnávací paměti v rezidentní části sdílené paměti. Další informace o použití vyrovnávacích pamětí s odlehčeným vstupem - výstupem naleznete v části “Odlehčený vstup - výstup pro inteligentní velké objekty” na stránce 5-20.

Parametr LOGBUFF

Parametr LOGBUFF ovlivňuje aktivitu protokolování vstupu - výstupu, protože určuje velikost vyrovnávacích pamětí logického protokolu ve sdílené paměti. Velikost těchto vyrovnávacích pamětí určuje, jak rychle se vyplní a jak často je proto potřeba je vyprázdnit na disk.

Pokud protokolujete uživatelská data inteligentních velkých objektů, zvětšete velikost vyrovnávací paměti logického protokolu, abyste zabránili častému vyprazdňování do těchto souborů protokolu na disku.

Volby obslužného programu onspaces, které ovlivňují vstup - výstup prostoru sbspace

Při vytváření prostoru sbspace můžete určit následující volby obslužného programu **onspaces**, které ovlivňují výkon vstupu - výstupu:

- Velikosti oblastí
- Režim ukládání do vyrovnávací paměti (použít odlehčený vstup - výstup, či nikoliv)
- Protokolování

Velikosti oblastí pro rozšíření prostoru sbspace

Jakmile přidáte inteligentní velké objekty do tabulky, databázový server přidělí prostorům sbspace místo na disku v jednotkách zvaných *oblasti*. Každá oblast je blok fyzicky souvislých stránek z prostoru sbspace. I když prostor sbspace obsahuje více než jeden blok, každá oblast je zcela přidělena jednomu bloku, takže zůstane souvislá.

Souvislost je důležitá pro výkon vstupu - výstupu. Když jsou stránky dat souvislé, pohyb ramena disku je při sekvenčním čtení řádek databázovým serverem snížen na minimum. Mechanismus oblastí je kompromisem mezi následujícími protichůdnými požadavky:

- Velikost některých inteligentních velkých objektů není předem známa.
- Počet inteligentních velkých objektů v rozdílných tabulkách může v různou dobu narůstat rozdílným tempem.
- Všechny stránky jednoho inteligentního velkého objektu by měly být v ideálním případě sousedící, aby bylo dosaženo nejlepšího výkonu při načtení celého objektu.

Protože pravděpodobně nejste schopni předpovědět počet a velikost inteligentních velkých objektů, nemůžete určit délku oblasti inteligentních velkých objektů. Proto přidává databázový server oblasti pouze tehdy, pokud jsou třeba, ale všechny stránky v jedné oblasti jsou souvislé, aby byl zajištěn vyšší výkon. Když navíc databázový server vytváří novou oblast, která přiléhá k předešlé oblasti, zachází s oběmi oblastmi jako s jednou.

Počet stránek v oblasti prostoru sbspace určuje jedna z následujících metod:

- Databázový server vypočítává velikost oblasti inteligentních velkých objektů na základě sady heuristických pravidel, využívá například údaj o počtu bajtů přenášených při operaci zápisu. Pokud například operace požaduje zápis 30 kB dat, pokusí se databázový server přidělit oblast o velikosti 30 kB.
- Výsledná velikost inteligentního velkého objektu tak, jak ukazuje jedna z následujících funkcí při otevření prostoru sbspace v aplikačním programu:

DB-Access

– Funkce DataBlade API **mi_lo_specset_estbytes**

Další informace o funkcích rozhraní DataBlade API pro otevření inteligentního velkého objektu a nastavení odhadovaného počtu bajtů naleznete v příručce *IBM Informix DataBlade API Programmer's Guide*.

Konec DB-Access

Jazyk ESQL/C

– Funkce ESQL/C **ifx_lo_specset_estbytes**

Další informace o funkcích jazyka ESQL/C pro otevření inteligentního velkého objektu a nastavení odhadovanéhopočtu bajtů naleznete v příručce *IBM Informix ESQL/C*

Programmer's Manual.

Konec Jazyk ESQL/C

Tyto funkce jsou nejlepším způsobem, jak nastavit velikost oblasti, protože snižují počet oblastí v inteligentním velkém objektu. Databázový server se pokusí přidělit celý inteligentní velký objekt jako jedinou oblast (pokud je v daném bloku dostupná oblast potřebné velikosti).

- Příznak EXTENT_SIZE ve volbě **-Df** příkazu **onspaces** při vytvoření nebo změně prostoru sbspace

  Většina administrátorů nepoužívá příznak EXTENT_SIZE obslužného programu **onspaces**, protože databázový server vypočítává velikost oblasti podle heuristických pravidel. Použití příznaku EXTENT_SIZE obslužného programu **onspaces** byste ale mohli zvážit v následujících situacích:

  – Mnoho jednostránkových oblastí je rozptýleno po celém prostoru sbspace.

  – Téměř všechny inteligentní velké objekty mají stejnou délku.

- Klíčové slovo EXTENT SIZE příkazu CREATE TABLE při určení sloupce CLOB nebo BLOB

  Většina administrátorů při vytváření nebo změně tabulky nepoužívá klíčové slovo EXTENT SIZE, protože databázový server vypočítává velikost oblasti podle heuristických pravidel. Mohli byste ale zvážit použití klíčové slova EXTENT SIZE v případě, kdy mají téměř všechny inteligentní velké objekty stejnou délku.

Důležité: Pro většinu aplikací je doporučeno používat velikost oblasti vypočtenou databázovým serverem. Funkci **mi_lo_specset_extsz** proměnné DataBlade API nebo funkci **ifx_lo_specset_extsz** jazyka ESQL/C nepoužívejte k nastavení velikosti oblasti pro rozšíření inteligentního velkého objektu.

V případě, že znáte velikost inteligentního velkého objektu, doporučuje se zadat velikost ve funkci rozhraní **DataBlade API mi_lo_specset_estbytes()** nebo ve funkci jazyka ESQL/C **ifx_lo_specset_estbytes()** a ne v obslužném programu **onspaces** nebo v příkazech CREATE TABLE a ALTER TABLE. Tyto funkce jsou nejlepším způsobem, jak nastavit velikost oblasti, protože databázový server přiděluje celý inteligentní velký objekt jako jedinou oblast (pokud je v daném bloku sousvislé úložiště).

Velikosti oblasti pro rozšíření větší než jeden megabajt neposkytují veliké zlepšení vstupu - výstupu, protože databázový server provádí čtecí a zapisovací operace maximálně v násobcích 60 kB. Databázový server ale registruje každou oblast pro inteligentní velký objekt v oblasti metadat, proto by hlavně velké inteligentní velké objekty mohly mít mnoho položek oblastí. Výkon databázového serveru by se při přístupu do těchto položek oblastí mohl zhoršit. V tomto případě můžete snížit počet položek oblastí v oblasti metadat tím, že určíte konečnou velikost inteligentního velkého objektu ve funkci **mi_lo_specset_estbytes()**nebo **ifx_lo_specset_estbytes()**.

Další informace naleznete v části “Zlepšení vstupu - výstupu metadat inteligentních velkých objektů” na stránce 6-13.

Odlehčený vstup - výstup pro inteligentní velké objekty

Inteligentní velké objekty se ve výchozím nastavení předávají skrz společnou oblast vyrovnávací paměti v rezidentní části sdílené paměti. Přestože mají inteligentní velké objekty nižší prioritu než ostatní data, společná oblast vyrovnávací paměti se může zaplnit, když aplikace přistupuje k mnoha inteligentním velkým objektům. Jediná aplikace může společnou oblast vyrovnávací paměti vyplnit inteligentními velkými objekty a ponechat tak málo místa pro data, které by jiné aplikace mohly potřebovat. Když navíc databázový server provádí

prohledávání mnoha stránek ve společné oblasti vyrovnávací paměti, mohlo by se zahlcení a kolize v důsledku přihlášování a odhlášování jednotlivých stránek stát kritickým místem.

Administrátor a programátor mají namísto použití společné oblasti vyrovnávací paměti možnost využít *odlehčený vstup - výstup*. Operace odlehčeného vstupu - výstupu používají soukromou vyrovnávací paměť v relaci společné oblasti virtuální části sdílené paměti.

Důležité: Soukromé vyrovnávací paměti používejte pouze tehdy, když během operací čtení a zápisu čtete nebo zapisujete inteligentní velké objekty větší než 8080 bajtů a přistupujete k nim zřídka. To znamená, že v případě nepříliš častého volání funkce čtení a zápisu, kdy dochází ke čtení velkého množství dat v jediném vyvolání funkce, může odlehčený vstup - výstup zlepšit výkon vstupu - výstupu.

Výhody odlehčeného vstupu - výstupu pro inteligentní velké objekty: Odlehčený vstup - výstup zajišťuje následující výhody:

- Přenáší větší bloky dat během jedné operace vstupu - výstupu

  Bloky těchto vstupů - výstupů můžou být velké pouze 60 kB. Aby to však byla pro databázový server při přenosu jen jedna operace vstupu - výstupu, musí být bajty sousední.
- Vyhýbá se zahlcení společné oblasti vyrovnávací paměti při čtení mnoha stránek.
- Zabraňuje, aby při čtení mnoha následných stránek pro inteligentní velké objekty byly často zpřístupňované stránky vytlačeny ze společné oblasti vyrovnávací paměti.

Když použijete pro inteligentní velké objekty vyrovnávací paměti s odlehčeným vstupem - výstupem, databázový server by mohl přečíst několik stránek jednou operací vstup - výstup. Jedna operace vstupu - výstupu čte v několika stránkách inteligentních velkých objektů, až do velikosti oblasti. Další informace o tom, kdy určit velikost oblasti, naleznete v části “Velikosti oblastí pro rozšíření prostoru sbspace” na stránce 5-19.

Určení odlehčeného vstupu - výstupu pro inteligentní velké objekty: Chce-li administrátor určit, aby se při vytvoření prostoru sbspace použil odlehčený vstup - výstup, může použít tag BUFFERING s volbou **-Df** v příkazu **onspaces -c -S**. Výchozí nastavení tagu BUFFERING je ON, což znamená, že se použije společná oblast vyrovnávací paměti. Režim použití vyrovnávací paměti, který určíte (nebo je výchozí v případě, že ho neurčíte) v příkazu **onspaces** je výchozím režimem použití vyrovnávací paměti pro všechny inteligentní velké objekty uložené v prostoru sbspace.

Důležité: Platí obecné pravidlo, podle kterého při vytváření prostoru sbspace neurčujte režim použití vyrovnávací paměti, pokud operace čtení a zápisu inteligentních velkých objektů nepracují s daty většími než 8080 kB. Pokud čtete nebo zapisujete krátké bloky dat, například o velikosti 2 kB nebo 4 kB, ponechte výchozí nastavení “buffering=ON”, abyste dosáhli zvýšení výkonu.

Programátoři můžou přepsat výchozí režim použití vyrovnávací paměti, když vytvoří, otevřou nebo upraví instanci inteligentní velký objekt s funkcemi DataBlade API a ESQL/C . Programovací rozhraní aplikací DataBlade API a ESQL/C zajišťuje, aby příznak LO_NOBUFFER povolil odlehčený vstup - výstup pro inteligentní velké objekty.

Důležité: Příznak LO_NOBUFFER používejte jen tehdy, když provádíte operace čtení nebo zápisu inteligentních velkých objektů větší než 8080 bajtů a přistupujete k nim jen zřídka. To znamená, že v případě nepříliš častého volání funkce čtení a zápisu, kdy dochází ke čtení velkého množství dat v jediném vyvolání funkce, může odlehčený vstup - výstup zlepšit výkon vstupu - výstupu.

Další informace o těchto příznacích a funkcích naleznete v příručce *IBM Informix DataBlade API Programmer's Guide* a *IBM Informix ESQL/C Programmer's Manual*.

Protokolování

Pokud se rozhodnete protokolovat všechny operace zápisu dat uložených v prostoru sbspace, zvýší se aktivita vstupu - výstupu logického protokolu a využití paměti. Další informace naleznete v části “Parametr LOGBUFF” na stránce 5-18.

Jak optický podsystém ovlivňuje výkon

Optický podsystém (Optical Subsystem) rozšiřuje možnosti úložiště databázového serveru pro inteligentní velké objekty (dat typu TEXT nebo BYTE) o optické podsystémy write-once-read-many (WORM). K ukládání počátečních stránek dat typu TEXT nebo BYTE požadovaných optickým podsystémem (Optical Subsystem) do vyrovnávací paměti používá databázový server mezipaměť. Mezipaměť je obecná paměťová oblast. Databázový server přidává do mezipaměti jednoduché velké objekty požadované libovolnou aplikací, dokud je v mezipaměti místo. Chce-li aplikace v mezipaměti uvolnit místo, musí uvolnit data typu TEXT nebo BYTE, které používá.

Výhoda podstatného zlepšení výkonu se projeví tehdy, když jsou data typu TEXT nebo BYTE načítána přímo do paměti namísto ukládání do vyrovnávací paměti na disku. Při používání optického podsystému (Optical Subsystem) je proto důležité, aby byla správně nastavena velikost mezipaměti. Celkové množství místa dostupného v mezipaměti určuje konfigurační parametr OPCACHEMAX. Aplikace oznámí, že vyžadují přístup do části mezipaměti, když nastavují proměnnou prostředí **INFORMIXOPCACHE**. Další informace naleznete v části “INFORMIXOPCACHE” na stránce 5-23.

Jednoduché velké objekty, které se zcela nevejdou do zbývajícího volného prostoru v mezipaměti jsou uloženy v prostoru blobspace, který je určen konfiguračním parametrem STAGEBLOB. Tato pracovní oblast se chová jako sekundární mezipaměť na disku pro stránky blobpage, které jsou načteny z optického podsystému (Optical Subsystem). Jednoduché velké objekty načtené z optického podsystému (Optical Subsystem) jsou drženy v pracovní oblasti, dokud nejsou dokončeny transakce, které si je vyžádaly.

Administrátor databázového serveru vytvoří prostor blobspace pracovní oblasti použitím jedné z následujících možností:

- Pomocí programu ON–Monitor (pouze systém UNIX)
- Pomocí obslužného programu ISA
- Pomocí obslužného programu **onspaces**

Chcete-li sledovat využití mezipaměti a prostoru blobspace STAGEBLOB, můžete použít **onstat -O** nebo ISA (**Výkon > Mezipaměť > Optická mezipaměť)** . V případě, že nastane spor mezipaměti, zvětšete hodnotu parametru OPCACHEMAX vypsanou v konfiguračním souboru. (Účinek nové hodnoty se projeví při dalším spuštění sdílené paměti databázovým serverem.) Úplný popis optického podsystému (Optical Subsystem) naleznete v příručce *IBM Informix Optical Subsystem Guide*.

Proměnné prostředí a konfigurační parametry pro optické podsystémy

Následující konfigurační parametry ovlivňují výkon optického podsystému (Optical Subsystem):

- STAGEBLOB
- OPCACHEMAX

Následující části tyto parametry popisují; popisují také proměnnou prostředí **INFORMIXOPCACHE** .

STAGEBLOB

Konfigurační parametr STAGEBLOB identifikuje daný prostor blobspace, který má být použitý jako pracovní oblast pro data typu TEXT nebo BYTE načtené z optického podsystému (Optical Subsystem) a optický podsystém (Optical Subsystem) aktivuje. V případě, že konfigurační soubor nevypíše parametr STAGEBLOB, optický podsystém (Optical Subsystem) nerozezná podsystém optického úložiště.

Struktura prostoru blobspace pracovní oblasti je stejná jako struktura všech ostatních prostorů blobspace databázových serverů. Když administrátor databázového serveru pracovní oblast vytvoří, skládá se pouze z jednoho bloku. Další bloky však lze přidat podle potřeby. Prostor blobspace pracovní oblasti nelze zrcadlit. Optimální velikost prostoru blobspace pracovní oblasti závisí na následujících faktorech:

- Frekvence úložiště jednoduchého inteligentního objektu.
- Frekvence vyhledávání jednoduchého inteligentního objektu.
- Průměrná velikost inteligentního velkého objektu k uložení.

Chcete-li vypočítat velikost prostoru blobspace pracovní oblasti, musíte odhadnout očekávaný počet jednoduchých velkých objektů, které v prostoru blobspace současně budou a musíte toto číslo vynásobit průměrnou velikostí jednoduchého inteligentního objektu.

OPCACHEMAX

Konfigurační parametr OPCACHEMAX určuje celkové množství místa dostupného pro vyhledání inteligentního velkého objektu v mezipaměti, kterou používá optický podsystém (Optical Subsystem). Dokud se mezipaměť nezaplní, ukládají se do ní jednoduché velké objekty požadované libovolnou aplikací. Jednoduché velké objekty, které se nevejdou do mezipaměti jsou uloženy na disk do prostoru blobspace, který označuje konfigurační parametr STAGEBLOB. Chcete-li omezit spory mezi požadavky inteligentních velkých objektů a zlepšit výkon požadavků, které vyžaduje optický podsystém (Optical Subsystem), zvětšete velikost mezipaměti.

INFORMIXOPCACHE

Proměnná prostředí **INFORMIXOPCACHE** nastavuje velikost mezipaměti, kterou daná aplikace používá k vyhledávání inteligentních velkých objektů. Pokud hodnota této proměnné přesáhne maximum zadané konfiguračním parametrem OPCACHEMAX, je místo ní použit parametr OPCACHEMAX. V případě, že není proměnná **INFORMIXOPCACHE** v prostředí určena, velikost mezipaměti je ve výchozím nastavení nastavena podle parametru OPCACHEMAX.

Vstup - výstup tabulky

Jednou z nejčastěji prováděných funkcí databázového serveru je přenášení dat a stránek rejstříků z disku do paměti. Pro krátké transakce můžou být stránky čteny jednotlivě, postupně jsou čteny při dotazech. Počet stránek přenášených databázovým serverem do paměti a časování požadavků na vstup - výstup pro sekvenční prohledávání lze konfigurovat. Lze také označit způsob, jakým má databázový server reagovat v případě, kdy dotaz požaduje data z dočasně nedostupného prostoru dbspace. Následující části popisují zmíněné metody čtení stránek.

Informace o vstupu - výstupu pro inteligentní velké objekty naleznete v části “Parametry, které ovlivňují vstup - výstup pro inteligentní velké objekty” na stránce 5-17.

Sekvenční prohledávání

Když databázový server provádí sekvenční prohledávání dat nebo stránek rejstříku, většinu čekací doby vstupu - výstupu způsobuje hledání vhodné počáteční stránky. Chcete-li dramaticky zlepšit výkon sekvenčního prohledávání, zaveďte počet sousedních stránek pro každou operaci vstupu - výstupu. Akce, při které se při sekvenčním prohledávání berou spolu s první stránkou také další stránky, se nazývá *dopředné čtení* (read-ahead).

Důležité je také časování operací vstup - výstup potřebných pro sekvenční prohledávání. Pokud musí jednotkový proces prohledávání čekat na přenesení další sady stránek poté, co projde každou dávkou, vznikne prodleva. Nejvyšší efektivitu zajistí sekvenčnímu prohledávání časování druhých a následujících požadavků na čtení tak, aby přenesly stránky dříve, než jich je potřeba. Počet stránek k přenesení a frekvence vstupu - výstupu dopředného čtení je závislá na dostupnosti místa v mezipamětech. Čtení napřed může zvýšit vyčistění stránek na nepřijatelnou úroveň, pokud je s každou dávkou přenášeno příliš mnoho stránek, nebo pokud jsou dávky přenášeny příliš často. Informace o tom, jak konfigurovat dopředné čtení, naleznete v části “Parametry RA_PAGES a RA_THRESHOLD” na stránce 5-25.

Odlehčené prohledávání

Za některých okolností se může databázový server vyhnout zahlcení společné oblasti vyrovnávací paměti, když provádí sekvenční prohledávání. Takové sekvenční prohledávání se označuje jako *odlehčené prohledávání*.

Výkonnostní výhody použití lehkých prohledávání namísto společných oblastí vyrovnávacích pamětí pro sekvenční prohledávání jsou následující:

- Přenáší větší bloky dat během jedné operace vstupu - výstupu

  Velikosti těchto bloků vstup - výstup jsou obvykle 64 nebo 128 kB. Potřebné informace k určení velikosti bloku vstupu - výstupu, kterou podporuje vaše platforma, naleznete v souboru poznámek k počítači.
- Vyhýbá se zahlcení společné oblasti vyrovnávací paměti při čtení mnoha stránek.
- Zabraňuje, aby při čtení mnoha sousedních stránek pro jeden dotaz DSS byly často zpřístupňované stránky vytlačeny ze společné oblasti vyrovnávací paměti.

Odlehčená prohledávání můžou být použita pouze pro sekvenční prohledávání velkých tabulek dat a jsou tím nejrychlejším prostředkem pro provádění těchto prohledávání. Odlehčená prohledávání nepoužívají tabulky systémového katalogu, tabulky o menší velikosti než společné oblasti vyrovnávací paměti a tabulky obsahující data typu varchar.

Lehká prohledávání se vyskytnou za následujících podmínek:

- Optimalizátor zvolí sekvenční prohledávání tabulky.
- Počet stránek v tabulce je větší než počet vyrovnávacích pamětí ve společné oblasti vyrovnávací paměti.
- Úroveň izolace nezíská zámek nebo sdílený zámek na tabulku:
  - Úroveň izolace neaktualizované čtení (obsahující neprotokolující databáze).
  - Úroveň izolace opakovatelné čtení, pokud má tabulka sdílený nebo výlučný zámek.
  - Izolace potvrzené čtení, pokud má tabulka sdílený zámek.

odlehčená prohledávání se nevyskytují při izolaci stability kurzoru.

Když databázový server používá konfigurační parametr RTO_SERVER_RESTART, odlehčená prohledávání automaticky nastaví příznak pro aktivaci dalšího kontrolního bodu.

Výstup obslužného programu **onstat -g lsc** ukazuje, kdy dojde k odlehčenému prohledávání, jak znázorňuje následující příklad výstupu. Příkaz **onstat -g lsc** zobrazí jen aktuálně aktivní odlehčená prohledávání.

```
Light Scan Info
descriptor  address  next_lpage  next_ppage  ppage_left  bufcnt look_aside
6           aaa7870 3f4         200429      1488        1      N
```

Nedostupná data

Další aspekt vstupu - výstupu tabulky má co do činění se situacemi, ve kterých dotaz požaduje přístup k tabulce nebo fragmentu v prostoru dbspace, který je dočasně nedostupný. Pokud databázový server určí, že je prostor dbspace nedostupný z důvodu selhání disku, dotazy směřované k tomuto prostoru dbspace se ve výchozím nastavení nezdaří. Databázový server umožňuje určit prostory dbspace, které můžou být v případě nedostupnosti dotazy přeskočeny, jak je popsáno v části “DATASKIP” na stránce 5-26.

Upozornění: Pokud je prostor dbspace obsahující data požadované dotazem vypsaný v parametru DATASKIP a je v současné době nedostupný z důvodů selhání disku, data, která databázový server vrátí dotazu, mohou být nekonzistentní se skutečným obsahem databáze.

Konfigurační parametry, které ovlivňují vstup - výstup tabulky

Následující konfigurační parametry ovlivňují dopředné čtení:

- RA_PAGES
- RA_THRESHOLD

Konfigurační parametr DATASKIP navíc povoluje a zakazuje přeskakování dat.

Následují části popisují účinky na výkonnost a úvahy spojované s těmito parametry. Další informace o konfiguračních parametrech databázového serveru naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Parametry RA_PAGES a RA_THRESHOLD

Konfigurační parametr RA_PAGES označuje počet stránek přenášených databázovým serverem do paměti pomocí jedné operace vstupu - výstupu během sekvenčního prohledávání dat nebo stránek rejstříku. Parametr RA_THRESHOLD označuje okamžik, ve kterém databázový server vydá požadavek vstup - výstup pro přenesení další sady stránek z disku. Protože je větší část čekací doby vstupu - výstupu způsobena hledáním správného začátečního bodu na disku, můžete zvýšit efektivitu sekvenčního prohledávání tím, že zvětšíte počet sousedních stránek přenášených při každém přenosu.

Nastavení příliš velkého počtu stránek u parametru RA_PAGES nebo nastavení příliš vysoké hodnoty parametru RA_THRESHOLD vzhledem k hodnotě vyrovnávací paměti v konfiguračním parametru BUFFERPOOL může spustit zbytečné vyčistění stránky, aby se vytvořilo místo pro stránky, které nejsou bezprostředně potřebné.

K výpočtu hodnot pro parametry RA_PAGES a RA_THRESHOLD použijte následující vzorce:

```
RA_PAGES = ((BUFFERS * bp_fract) / (2 * large_queries)) + 2
RA_THRESHOLD = ((BUFFERS * bp_fract) / (2 * large_queries)) - 2
```

bp_fract je část vyrovnávací paměti dat, která se použije pro velké prohledávání vyžadující dopředné čtení. Pokud chcete umožnit, aby velké prohledávání zabralo až 75 procent vyrovnávacích pamětí, hodnota *bp_fract* bude 0.75.

large_queries je počet souběžných dotazů vyžadujících dopředné čtení, které máte v úmyslu podporovat.

DATASKIP

Konfigurační parametr DATASKIP umožňuje určit, které prostory dbspace, pokud nějaké, můžou být dotazy přeskočeny v případě, že jsou tyto prostory nedostupné z důvodu selhání disku. Můžete vypsat určité prostory dbspace a zapnout nebo vypnout možnost přeskakování dat pro všechny prostory dbspace. Podrobnější informace naleznete v příručce *IBM Informix Administrator's Guide*

V případě, že je přeskakování dat povoleno, databázový server nastaví šestý znak v poli SQLWARN na W. Další informace o poli SQLWARN naleznete v příručce *IBM Informix Guide to SQL: Tutorial*.

Upozornění: Když je nějaký prostor dbspace přeskočený, databázový server nedokáže určit, zda jsou výsledky dotazu konzistentní. Pokud prostor dbspace obsahuje fragment tabulky, uživatel, který dotaz provádí, musí zajistit, aby řádky v tomto fragmentu nebyly potřebné pro přesný výsledek dotazu. Zapnutí parametru DATASKIP umožní, aby dotazy s nekompletními daty vrátily výsledky, které můžou být nekonzistentní se skutečným stavem databáze. Bez patřičné pozornosti ale tato data můžou přinést chybné nebo zavádějící výsledky dotazu.

Aktivity vstupu - výstupu na pozadí

Aktivity vstupu - výstupu na pozadí neprovádějí dotazy SQL okamžitě. Mnohé z těchto aktivit jsou nezbytné pro zachování konzistence databáze a dalších aspektů operací databázového serveru. Vytvářejí však režii v CPU a zabírají šířku pásma vstupu - výstupu. Režijní aktivity pak ubírají čas dotazům a transakcím. Pokud řádně nenakonfigurujete aktivity vstupu - výstupu na pozadí, příliš velká režie pro tyto aktivity může omezit propustnost transakcí vašich aplikací.

Následující seznam zobrazuje některé aktivity vstupu - výstupu na pozadí:

- Kontrolní body
- Protokolování
- Vyčištění stránky
- Zálohování a obnovení
- Odvolání a obnovení
- Replikace dat
- Auditování

Kontrolní body se vyskytnou nezávisle na tom, zda je velká aktivita databáze; s rostoucí aktivitou se však můžou vyskytovat častěji. Další aktivity na pozadí, jako například protokolování a vyčistění stránky, se vyskytují mnohem častěji, když roste využití databáze. Aktivity typu zálohování, obnovení nebo rychlá obnova se vyskytnou pouze tehdy, když jsou naplánované, nebo za výjimečných okolností.

Vyladění aktivit vstupu - výstupu na pozadí spočívá hlavně ve správném vyvážení mezi vhodnými intervaly kontrolních bodů, režimy protokolování, velikostmi protokolů a hodnotami prahů pro vyčistění stránky. Prahové hodnoty a intervaly spouštějící aktivity vstupu - výstupu na pozadí se často vzájemně ovlivňují; změnou jedné prahové hodnoty by se mohlo kritické místo pro výkon přesunout na jinou.

Následující části popisují účinky na výkonnost a úvahy spojované s těmito parametry, které ovlivňují aktivity vstupu - výstupu na pozadí. Další informace o konfiguračních parametrech databázového serveru naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Konfigurační parametry, které ovlivňují kontrolní body

Následující konfigurační parametry ovlivňují kontrolní body:

- RTO_SERVER_RESTART
- CKPTINTVL
- LOGSIZE
- LOGFILES
- PHYSFILE
- ONDBSPACEDOWN

RTO_SERVER_RESTART

Konfigurační parametr RTO_SERVER_RESTART použijte k nastavení množství času v sekundách, během kterého se dynamický server musí obnovit po neplánovaném výpadku.

Povolení tohoto konfiguračního parametru přináší následující výkonnostní výhodu:

- Umožňuje, aby rychlá obnova vyhověla zásadě RTO_SERVER_RESTART tím, že do společné oblasti vyrovnávací paměti umístí datové stránky vyžadované opakováním protokolu.

Povolení tohoto konfiguračního parametru přináší následující výkonnostní nevýhody:

- Zvýšená aktivita fyzického protokolu, která může nepatrně ovlivnit výkon transakcí.
- Zvýšená frekvence kontrolních bodů, protože dochází k rychlejšímu vyčerpání místa ve fyzickém protokolu. (Zvýšení frekvence kontrolních bodů můžete zabránit zvýšením velikosti fyzického protokolu.)

Pokud je parametr RTO_SERVER_RESTART povolený, databázový server provádí následující akce:

- Pokusí se zajistit, aby během zpracování neblokujících kontrolních bloků nedošly kritické zdroje tak, že spustí častější kontrolní body v případě, že by transakce mohly spotřebovat zdroje buď fyzického nebo logického protokolu, což by mohlo způsobit zablokování transakce.
- Ignoruje konfigurační parametr CKPTINTVL.
- Automaticky řídí frekvenci kontrolních bodů tak, aby splnil zásadu RTO a aby zabránil tomu, že databázový server spotřebuje zdroje protokolu.
- Automaticky upraví počet virtuálních procesorů AIO a jednotkových procesů vyčištění a automaticky vyladí vyprázdnění LRU.

Pokud je parametr RTO_SERVER_RESTART povolený, provádí databázový server také tyto akce:

Pokud server nedokáže splnit zásady RTO_SERVER_RESTART, databázový server vytiskne varovné zprávy do protokolu zpráv.

Automatické kontrolní body, ladění LRU a ladění virtuálního procesoru AIO:
Databázový server automaticky upravuje frekvenci kontrolních bodů, aby se vyhnul zablokování transakce. Aby toho server dosáhl, sleduje spotřebu fyzického a logického protokolu spolu s údaji o předešlém výkonu transakce. V případě potřeby potom server spustí kontrolní body častěji, aby se zablokování transakce vyhnul.

Automatické ladění kontrolního bodu můžete vypnout nastavením **onmode -wf AUTO_CKPTS** na hodnotu 0, nebo nastavením konfiguračního parametru AUTO_CKPTS na hodnotu 0.

Protože databázový server neblokuje transakce během zpracovávání kontrolních bodů, mělo by být vyprázdnění LRU uvolněno. Pokud server není schopen dokončit zpracování kontrolních bodů předtím, než dojde ke spotřebování fyzického protokolu (což způsobí zablokování transakce) a pokud nelze zvětšit velikost fyzického protokolu, můžete nakonfigurovat server, aby prováděl agresivnější vyprazdňování LRU. Zvýšení vyprazdňování LRU ovlivní výkon transakce, mělo by ale blokování transakce omezit. Pokud nenakonfigurujete server, aby prováděl agresivnější vyprazdňování, server automaticky upraví vyprazdňování LRU tak, aby bylo agresivnější jen v tom případě, že server není schopen najít vyrovnávací paměť s nízkou prioritou pro náhradu stránky.

Když je konfigurační parametr AUTO_AIOVPS povolený, databázový server automaticky zvýší počet virtuálních procesorů AIO a jednotkových procesů čištění stránek, když zjistí, že virtuální procesory AIO nestačí pracovnímu zatížení vstupu - výstupu.

Automatické ladění LRU ovlivňuje všechny společné oblasti vyrovnávací paměti a upravuje hodnoty **lru_min_dirty** a **lru_max_dirty** v konfiguračním parametru BUFFERPOOL. Další informace o tomto tématu a informace o tom, jak vypnout ladění LRU naleznete v části “Ladění LRU” na stránce 5-40.

CKPTINTVL

Pokud je konfigurační parametr RTO_SERVER_RESTART zapnutý, databázový server ignoruje konfigurační parametr CKPTINTVL. Místo toho server automaticky spustí kontrolní body, aby udržel zásady RTO_SERVER_RESTART.

Pokud není konfigurační parametr RTO_SERVER_RESTART zapnutý, určuje konfigurační parametr CKPTINTVL frekvenci (v sekundách), s jakou databázový server ověřuje, zda je potřebný kontrolní bod. Databázový server může kontrolní bod vynechat, pokud jsou v okamžiku, kdy interval kontrolního bodu uplyne všechna data fyzicky konzistentní.

Ke kontrolnímu bodu také dojde, kdykoliv bude fyzický protokol zaplněný ze 75 procent.

Pokud nastavíte dlouhý interval parametru CKPTINTVL, můžete použít kapacity fyzického protokolu, abyste spustili kontrolní body vycházející z aktivity databáze namísto z libovolné časové jednotky. Dlouhý časový interval ale zvyšuje dobu potřebnou pro obnovení v případě selhání. V závislosti na propustnosti a požadavcích na dostupnost dat si můžete zvolit počáteční interval kontrolního bodu mezi hodnotami 5, 10 a 15 minut. Je však potřeba vědět, že by se kontrolní body mohly vyskytnout častěji, a to v závislosti na aktivitě fyzického protokolování.

Databázový server zapíše zprávu do protokolu zpráv, aby zaznamenal časový okamžik dokončení kontrolního bodu. Chcete-li si tyto zprávy přečíst, použijte **onstat -m**.

LOGSIZE a LOGFILES

Konfigurační parametry LOGSIZE a LOGFILES nepřímo ovlivňují kontrolní body, protože určují velikost a počet souborů logického protokolu. Kontrolní bod se může vyskytnout, když databázový server zjistí, že v pořadí následující aktuální soubor logického protokolu obsahuje záznam posledního kontrolního bodu. Pokud potřebujete uvolnit soubor logického protokolu, který obsahuje poslední kontrolní bod, databázový server musí do aktuálního souboru logického protokolu zapsat nový záznam kontrolního bodu. Pokud se zvýší frekvence zálohování a uvolnění souborů logického protokolu, zvýší se také frekvence výskytu kontrolních bodů. Přestože kontrolní body blokují práci uživatele, už nebudou nadále trvat tak

dlouho. Tento vliv by však nemusel být důležitý, protože frekvenci kontrolního bodu určují také další faktory (jako například velikost fyzického protokolu).

Když je povolena funkce dynamického přidělení protokolu, velikost logického protokolu neovlivňuje prahové hodnoty dlouhých transakcí tak moc, jako v minulých verzích databázového serveru. Podrobnější informace naleznete v části “LTXHWM a LTXEHWM” na stránce 5-34.

Parametry LOGSIZE, LOGFILES a LOGBUFF ovlivňují také aktivitu protokolování vstupu - výstupu a logické zálohy. Další informace naleznete v části “Konfigurační parametry, které ovlivňují protokolování” na stránce 5-30.

PHYSFILE

Konfigurační parametr PHYSFILE použijte pro nastavení velikosti počátečního fyzického protokolu. Poté, co inicializujete diskový prostor a uvedete databázový server do stavu online pomocí příkazu **oninit -i**, použijte obslužný program **onparams** a změňte umístění a velikost fyzického protokolu.

Kontrolní bod se vyskytne, kdykoliv bude fyzický protokol zaplněný ze 75 procent.

Protože je fyzický protokol důležitou částí dodržení zásad RTO_SERVER_RESTART, měli byste mít velké množství prostoru fyzického protokolu. Velikost fyzického protokolu by měla být nejméně 110 procent velikosti všech společných oblastí vyrovnávací pamětí.

Fyzický protokol můžete změnit, když jsou transakce aktivní a nemusíte restartovat databázový server. Informace o odhadování velikost fyzického protokolu naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Jak kontrolní body ovlivňují fyzický protokol: Kontrolní body ovlivňují fyzický protokol. Během analýzy fyzického protokolu uvažujte také následující problémy:

- Jak rychle se fyzický protokol vyplňuje

  Operace, které neprovádějí aktualizace nevytvářejí předobrazy. Pokud se velikost databáze zvětšuje ale aplikace aktualizují data jen zřídka, mnoho fyzického protokolování se nevyskytuje. Za této situace velký fyzický protokol pravděpodobně nepotřebujete.

  Databázový server zapisuje předobraz pouze první aktualizace provedené ve stránce, například, pro následující operace:

  – Vložení, aktualizace a odstranění týkající se řádků obsahujících datové typy definované uživatelem, inteligentní velké objekty a jednoduché velké objekty.
  – Příkazy ALTER.
  – Operace vytvářející nebo měnící indexy (B-stromy, R-stromy nebo indexy definované uživatelem).

  Pokud aplikace aktualizuje stejné stránky, můžete definovat menší fyzický protokol.

- Jak často se kontrolní body vyskytují

  Protože je fyzický protokol po každém kontrolním bodu recyklován, musí být dostatečně velký na to, aby obsáhnul předobrazy ze změn mezi kontrolními body. Pokud databázový server často spouští kontrolní body, protože už nemá prostor ve fyzickém protokolu, zvažte možnost zvětšení velikosti fyzického protokolu.

- Kdy zvětšit velikost fyzického protokolu

  Zvětšit velikost fyzického protokolu možná budete chtít v případě, kdy zvětšíte interval kontrolního bodu nebo když očekáváte zvýšenou aktivitu aktualizace.

Informace o parametru PHYSFILE naleznete v kapitole týkající se konfiguračních parametrů v příručce *IBM Informix Dynamic Server Administrator's Reference*. Informace o fyzickém protokolu a kontrolních bodech naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

ONDBSPACEDOWN

Konfigurační parametr ONDBSPACEDOWN určuje odezvu, kterou databázový server provede, když chyba vstupu - výstupu ukazuje na nefunkční prostor dbspace. Databázový server ve výchozím nastavení označí všechny prostory dbspace neobsahující žádná kritická data jako nefunkční a pokračuje dál ve zpracování. Mezi kritická data patří kořenový prostor dbspace, logický protokol nebo fyzický protokol. Chcete-li obnovit do takové databáze, je potřeba zálohovat všechny logické protokoly a provést teplé obnovení nefunkčního prostoru dbspace.

Databázový server zastaví činnost bez ohledu na nastavení parametru ONDBSPACEDOWN, kdykoliv dojde k chybě zakázání vstupu - výstupu v nezrcadleném prostoru dbspace, který obsahuje kritická data. Když nastane tato událost, je potřeba provést studené obnovení databázového serveru, aby mohl pokračovat v běžné databázové činnosti.

Hodnota parametru ONDBSPACEDOWN nemá žádný vliv na dočasné prostory dbspace. U dočasných prostorů dbspace databázový server pokračuje ve zpracování bez ohledu na nastavení parametru ONDBSPACEDOWN. Pokud dočasný prostor dbspace vyžaduje opravu, můžete jej vypustit a znovu vytvořit.

Když je parametr ONDBSPACEDOWN nastavený na hodnotu 2, databázový server pokračuje ve zpracování až k dalšímu kontrolnímu bodu a potom pozastaví zpracování všech dotazů na aktualizaci. Databázový server opakuje požadavek na vstup - výstup, který vyvolal danou chybu, dokud se prostor dbspace neopraví a požadavek se nedokončí, nebo nezasáhne administrátor databázového serveru. Administrátor může k označení daného prostoru dbspace jako nefunkční použít **onmode -O** a pokračovat ve zpracování, zatímco je daný prostor nedostupný, nebo může použít **onmode -k**, aby zastavil databázový server.

Důležité: Nastavení hodnoty u parametru ONDBSPACEDOWN na *2* může vážně ovlivnit výkon požadavků na aktualizaci, protože ty jsou pozastaveny kvůli nefunkčnímu prostoru dbspace. Při použití tohoto nastavení pro parametr ONDBSPACEDOWN určitě sledujte stav prostorů dbspace.

Pokud nastavíte parametr ONDBSPACEDOWN na hodnotu 1, bude databázový server zacházet se všemi prostory dbspace jako by byly kritické. Každá nezrcadlený prostor dbspace, který se stane nepřístupným zastaví běžné zpracování a bude vyžadovat studené obnovení. Dopad na výkon v případě zastavení a provedení studeného obnovení, když se prostor dbspace stane nefunkčním, může být vážný.

Důležité: Pokud se rozhodnete nastavit hodnotu parametru ONDBSPACEDOWN na *1*, zvažte možnost zrcadlení všech prostorů dbspace.

Konfigurační parametry, které ovlivňují protokolování

Kontrolní body, protokolování a vyčistění stránek jsou pro udržení konzistence databáze nezbytné. Mezi frekvencí kontrolních bodů nebo velikostí logických protokolů a mezi časem potřebným pro obnovu databáze v případě selhání existuje přímý vztah. Nejdůležitějším předmětem úvahy se tedy při pokusu o snížení zatížení těchto aktivit stane velikost prodlevy, která je v průběhu obnovy přijatelná.

Protokolování ovlivňují následující konfigurační parametry:

- LOGBUFF

- PHYSBUFF
- LOGFILES
- LOGSIZE
- DYNAMIC_ LOGS
- LTXHWM
- LTXEHWM
- TEMPTAB_NOLOG

LOGBUFF a PHYSBUFF

Konfigurační parametry LOGBUFF a PHYSBUFF ovlivňují aktivitu protokolování vstupu - výstupu, protože určují příslušné velikosti vyrovnávacích pamětí logického a fyzického protokolu ve sdílené paměti. Velikost těchto vyrovnávacích pamětí určuje, jak rychle se vyplní a jak často je proto potřeba je vyprázdnit na disk.

LOGFILES

Parametr LOGFILES určuje počet souborů logického protokolu.

Odhad počtu souborů logického protokolu: Pokud mají všechny soubory logického protokolu stejnou velikost, můžete vypočítat celkový prostor přidělený souborům logického protokolu následujícím způsobem:

```
celkový prostor logického protokolu = LOGFILES * LOGSIZE
```

Pokud přidáte soubory logického protokolu, které neodpovídají velikosti určené parametrem LOGSIZE, nemůžete k výpočtu velikosti logického protokolu použít výše uvedený výraz LOGFILES * LOGSIZE. Místo toho musíte sečíst velikosti jednotlivých souborů protokolu na disku.

Ke sledování souborů logického protokolu použijte obslužný program **onstat -l**.

LOGSIZE

Parametr LOGSIZE použijte pro nastavení velikosti každého souboru logického protokolu. Dokud není systém plně v provozu, je těžké předpovědět, kolik prostoru pro logický protokol bude systém databázového serveru vyžadovat.

Velikost prostoru logického protokolu (LOGFILES * LOGSIZE) určují tyto zásady:

Cílová doba obnovení (RTO)

Doba, po kterou si můžete dovolit být bez vašich systémů. Pokud je vaším jediným cílem obnova po selhání, celkový prostor logického protokolu potřebuje být veliký pouze tak, aby obsahoval všechny transakce během dvou cyklů kontrolního bodu. Pokud je konfigurační parametr RTO_SERVER_RESTART zapnutý a server má kombinovanou velikost společné oblasti vyrovnávací paměti menší než 4 GB, můžete konfigurovat celkovou velikost prostoru protokolu na 110 % kombinovaných velikostí společné oblasti vyrovnávací paměti. Příliš mnoho prostoru protokolu výkon neovlivní; příliš málo prostoru protokolu může ale způsobit častější kontrolní body a blokování transakcí.

Cílový bod pro obnovu (RPO)

Popisuje stáří dat, které chcete obnovit v případě nehody. Pokud je cílem zajištění ochrany transakční práce, optimální hodnota parametru LOGSIZE by měla být násobkem množství vykonané práce za jednotku RPO. Protože databázový server podporuje částečné zálohování protokolu, optimální velikost prostoru protokolu není kritická a neoptimální velikost prostoru protokolu znamená pouze častější změny v souboru protokolu. RPO se měří v časových jednotkách. Pokud je obchodním

pravidlem, že systém nesmí ztratit více než deset minut transakčních dat v případě celkové havárie serveru, pak by se mělo zálohování protokolu provést každých deset minut.

Pro nastavení automatického zálohování protokolu můžete použít plánovač, který bude spravovat a provádět naplánované úlohy administrace. Informace o plánovači naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Dlouhé transakce
Pokud máte dlouhé transakce, které vyžadují velké množství prostoru protokolu, měli byste toto množství prostoru protokolům přidělit. Nedostatečné množství prostoru protokolu ovlivní výkon transakce.

Velikost protokolu zvolte podle množství výskytu aktivity protokolování a podle množství rizika v případě katastrofálního selhání. Pokud si nemůžete dovolit ztratit více než hodinovou hodnotu dat, vytvořte si hodně malých souborů protokolu, z nichž každý bude uchovávat hodinovou hodnotu transakcí. Zapněte nepřetržité zálohování protokolu. Malé soubory logického protokolu se vyplní dříve, což znamená častější zálohování logického protokolu.

Pokud je systém stabilní při vysoké aktivitě protokolování, zvolení větších souborů protokolů zlepší výkon. U větších souborů protokolu se nepřetržité zálohování protokolu provádí méně často. Zvažte také maximální míru transakcí a rychlost zálohovacích zařízení. Nedovolte úplné zaplnění logického protokolu. Chcete-li se vypořádat i s nejdelšími transakcemi, zapněte nepřetržité zálohování protokolu a ponechte v logickém protokolu dostatek volného prostoru.

Proces zálohování může bránit zpracování transakcí, které používá data umístěná na stejném disku jako soubory logického protokolu. Pokud je k dispozici dostatek volného prostoru logického protokolu na disku, můžete počkat na okamžiky nízké aktivity uživatele předtím, než zálohujete soubory logického protokolu.

Odhad velikosti logického protokolu při protokolování prostorů dbspace: Chcete-li získat počáteční odhad velikosti parametru LOGSIZE v kilobajtech, můžete použít následující vzorec:

```
LOGSIZE = (connections * maxrows * rowsize) / 1024) / LOGFILES
```

connections Určuje maximální počet připojení pro všechny typy sítě zadané v souboru nebo registru **sqlhosts** jedním nebo více parametry NETTYPE. Pokud jste v konfiguračním souboru nakonfigurovali více než jedno připojení nastavením několika konfiguračních parametrů NETTYPE, sečtěte pole **users** pro každý parametr NETTYPE a tímto součtem nahraďte hodnotu *connections* v předchozím vzorci.

maxrows je největší počet řádků, který bude aktualizován jedinou transakcí.

rowsize je průměrná velikost řádku tabulky v bajtech. Hodnotu *rowsize* můžete vypočítat tak, že přičtete délku sloupců v řádku (z tabulky systémového katalogu **syscolumns**.

1024 je nezbytný dělitel, protože parametr LOGSIZE určujete v kilobajtech.

Chcete-li získat lepší odhad, v okamžicích nejvyšší aktivity proveďte příkaz **onstat -u**. Poslední řádek výstupu příkazu **onstat -u** obsahuje maximální počet souběžných připojení.

Pokud vaše transakce zahrnují jednoduché velké objekty nebo inteligentní velké objekty, potřebujete upravit velikost logického protokolu, jak popisují následující části.

Můžete také zvýšit množství prostoru věnovaného logickému protokolu tím, že přidáte další soubor logického protokolu, jak je vysvětleno v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Odhad velikosti logického protokolu při protokolování jednoduchých velkých objektů: Pokud chcete dosáhnout lepšího celkového výkonu aplikací, které provádějí časté aktualizace dat typu TEXT a BYTE v prostorech blobspace, zmenšete velikost logického protokolu. Stránky blobpage nemůžou být znova použity, dokud není zálohovaný logický protokol, do kterého jsou přiděleny. Když je aktivita dat typu TEXT nebo BYTE vysoká, vliv častějších kontrolních bodů na výkon je vyvážený větší dostupností volných stránek blobpage.

Pokud používáte v prostorech blobspace přechodné stránky blobpage, menší protokoly mohou zlepšit přístup k jednoduchým velkým objektům, které musí být používány opakovaně. Jednoduché velké objekty nemohou být znova použity, dokud není logický protokol, do kterého jsou přiděleny, vyprázdněn na disk. V tomto případě lze ospravedlnit úbytek výkonu, protože tyto menší soubory protokolu jsou zálohovány častěji.

Odhad velikosti logického protokolu při protokolování inteligentních velkých objektů (IDS): Pokud plánujete protokolovat uživatelská data inteligentních velkých objektů, musíte zajistit, aby velikost protokolu byla značně větší, než objem zapisovaných dat. Metadata inteligentních velkých objektů jsou protokolována vždy, i když inteligentní velké objekty protokolovány nejsou.

Když protokolujete inteligentní velké objekty, použijte následující pravidla:

- Pokud dodáváte data k inteligentnímu velkému objektu, zvýšená aktivita protokolování je přibližně rovna množství dat zapsaných do inteligentního velkého objektu.
- Pokud aktualizujete inteligentní velký objekt (přepisujete data), zvýšená aktivita protokolování je přibližně dvakrát tak větší než množství dat zapsaných do inteligentního velkého objektu. Databázový server protokoluje předobraz a after-image inteligentního velkého objektu pro aktualizační transakce. Při aktualizaci inteligentních velkých objektů databázový server protokoluje pouze aktualizované části předobrazu a after image.
- Aktualizace metadat mají na protokolování menší vliv. I když jsou metadata vždy protokolována, počet protokolovaných bajtů je obvykle mnohem menší než inteligentní velké objekty.

DYNAMIC_LOGS

Funkce dynamického přidělení logického protokolu zabraňuje zhroucení způsobené odvoláním dlouhé transakce, protože databázový server nespotřebuje prostor protokolu. Dynamické přidělování protokolů umožňuje provádět následující činnosti:

- Přidat soubor logického protokolu, zatímco je systém aktivní, dokonce i během rychlého obnovení.
- Vložit soubor logického protokolu ihned za soubor současného protokolu namísto jeho přidání na konec.
- Okamžitě přistupovat k souborům logického protokolu, přestože kořenový prostor dbspace nebyl zálohován.

Výchozí hodnotou konfiguračního parametru DYNAMIC_LOGS je 2, což znamená, že databázový server automaticky přidělí nový soubor logického protokolu za aktuální soubor protokolu tehdy, pokud zjistí, že následující soubor protokolu obsahuje otevřenou transakci. Databázový server automatický zkontroluje, zda protokol za aktuálním protokolem stále obsahuje otevřenou transakci v následujících okamžicích:

- Ihned poté, co se přepne do nového souboru protokolu během zapisování záznamů protokolu (ne během čtení a zpracováním záznamů protokolu).

- Na začátku fáze vyčištění transakcí, která se vyskytne jako poslední fáze při logické obnově.

  Logická obnova se provede na konci rychlé obnovy a na konci studeného obnovení nebo přehrání protokolu. Další informace o fázích rychlé obnovy naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.
- V průběhu vyčištění transakce (odvolání otevřené transakce) by se mohlo vyskytnout přepnutí do nového souboru protokolu.

  Po tomto přepnutí databázový server rovněž provádí kontrolu, protože přepnutí zapisuje záznamy protokolu (z důvodu?) odvolání.

Pokud pro parametr DYNAMIC_LOGS použijete výchozí hodnotu 2 , databázový server určí umístění a velikost nového souboru logického protokolu:

- Databázový server používá při určení disku, na který přidělí nový soubor protokolu následující podmínky:
  - Upřednostňuje zrcadlení prostory dbspace.
  - Vyhýbá se kořenovému prostoru dbspace, dokud jsou dostupné jiné kritické prostory dbspace.
  - Nejméně upřednostňovanými prostory jsou nezrcadlené a nekritické prostory dbspace.
- Databázový server používá k určení velikosti nového souboru logického protokolu průměrnou velikost největšího a nejmenšího souboru protokolu. Pokud pro tuto průměrnou velikost není dostatek souvislého prostoru na disku, databázový server vyhledá prostor pro další nejmenší průměrnou velikost. Databázový server pro nový soubor protokolu přidělí minimálně 200 kB.

Pokud chcete ovládat umístění a velikost dodatečného souboru protokolu, nastavte hodnotu parametru DYNAMIC_LOGS na 1. Kontrolu toho, zda následující aktivní protokol neobsahuje aktivní transakci provádí databázový server i při přepínání souborů protokolu. Pokud v následujícím protokolu, který se má stát aktivním, otevřenou transakci nalezne, provede následující akce:

- Vyvolá akci při varování 27 (vyžadován protokol).
- Zapíše varovnou zprávu do online protokolu.
- Zastaví se a počká, až administrátor ručně přidá protokol pomocí volby příkazové řádky **onparams -a -i**.

Lze vytvořit skript, který při výskytu akce při varování 27 provede příkaz **onparams -a -i** s umístěním, které chcete použít pro nový protokol. Skript také může provádět příkaz **onstat -d** , aby vyhledal odpovídající prostor a provedl příkaz **onparams -a -i** s umístěním, ve kterém se nachází dostatek prostoru. Chcete-li přidat nový protokol přímo za aktuální soubor protokolu, musíte použít volbu **-i**.

Pokud nastavíte hodnotu parametru DYNAMIC_LOGS na 0, databázový server bude při přepínání souborů protokolu stále kontrolovat, zda následující aktivní protokol neobsahuje otevřenou transakci. Pokud v následujícím protokolu, který se má stát aktivním, otevřenou transakci nalezne, vydá následující upozornění:

```
Upozornění: Nejstarší soubor logického protokolu (%d) obsahuje záznamy
otevřené transakce (0x%p), ale funkce souborů dynamického protokolu
je vypnuta.
```

LTXHWM a LTXEHWM

Díky funkci dynamického protokolování souboru už nejsou vysoké meze dlouhé transakce tak kritické, jak tomu bývalo v dřívějších verzích před verzí 9.3, protože databázový server nespotřebuje všechen prostor protokolu, pokud nevyčerpáte fyzické místo na disku dostupné

v systému. Proto verze 9.4 nemá parametry LTXHWM a LTXEHWM v souboru **onconfig.std** a tyto parametry mají následující výchozí hodnoty v závislosti na hodnotě konfiguračního parametru DYNAMIC_LOGS:

- S přidělením souboru dynamického protokolu

  Když použijete pro parametr DYNAMIC_LOGS výchozí hodnotu 2 nebo nastavíte hodnotu parametru DYNAMIC_LOGS na 1, výchozí hodnoty vysokých mezí dlouhých transakcí budou:

  ```
  LTXHWM 80
  LTXEHWM 90
  ```

  Výchozí hodnoty parametrů LTXHWM a LTXEHWM jsou větší než v předchozí verzi, protože databázový server přidává logické protokoly bez nutnosti restarování. Protože databázový server nespotřebuje všechny protokoly, jiní uživatelé můžou během odvolání dlouhé transakce do protokolu stále přistupovat.

- Bez přidělení souboru dynamického protokolu

  Pokud nastavíte hodnotu parametru DYNAMIC_LOGS na 0, výchozí hodnota parametru LTXHWM bude 50 a výchozí hodnota parametru LTXEHWM bude 60 .

Když databázový server začne kontrolovat možnou dlouhou transakci a odvolá ji, parametr LTXHWM stále ukazuje zaplněnost logického protokolu. Parametr LTXEHWM stále označuje bod, ve kterém databázový server pozastaví aktivitu nové transakce, aby vyhledal a odvolal dlouhou transakci. K těmto událostem by mělo docházet výjimečně, pokud se ale vyskytnou, může to naznačovat vážnější problém v aplikaci.

Důležité: Výchozí hodnoty parametrů LTXHWM a LTXEHWM se doporučuje zachovat.

Při běžné činnosti použijte pro pro parametry LTXHWM a LTXEHWM výchozí hodnoty. Výchozí hodnoty však možná budete chtít změnit v jednom z následující důvodů:

- Abyste během odvolání dlouhé transakce umožnili jiným transakcím pokračovat v aktualizační aktivitě (což vyžaduje přístup do protokolu).

  V tomto případě zvětšete hodnotu parametru LTXEHWM, aby se zvýšil bod, ve kterém má odvolání transakce výlučný přístup do protokolu.

- Aby se provedly naplánované transakce neznámých délek, jako například velká načtení, která jsou protokolována.

  V tomto případě zvětšete hodnotu parametru LTXEHWM, aby transakce měla příležitost se dokončit dříve než dosáhne vysoké meze.

TEMPTAB_NOLOG

Konfigurační parametr TEMPTAB_NOLOG umožňuje zakázat protokolování dočasných tabulek. To umožňuje zlepšit výkon a zabránit serveru Dynamic Server v přenášení dočasných tabulek při použití replikace HDR.

Chcete-li zakázat protokolování dočasných tabulek, nastavte konfigurační parametr TEMPTAB_NOLOG na hodnotu 1.

Konfigurační parametry, které ovlivňují vyčištění stránky

Pokud nejsou stránky vyčištěny dostatečně často, jednotkový proces **sqlexec** provádějící dotaz by nemusel být schopen nalézt dostupné stránky, které potřebuje. Musí potom inicializovat *zápis na popředí* a čekat na uvolnění stránek. Zápisy na popředí zhoršují výkon, takže byste se jim měli vyhnout. Chcete-li snížit frekvenci zápisu na popředí, zvětšete počet čističů stránek, nebo snižte prahovou hodnotu pro spuštění vyčištění stránky.

Ke sledování frekvence zápisu na popředí použijte volbu **onstat -F**.

Následující konfigurační parametry ovlivňují vyčištění stránky:

- BUFFERPOOL, který obsahuje hodnoty **lrus**, **lru_max_dirty** a **lru_min_dirty**.

  Informace, které byly určovány pomocí konfiguračních parametrů BUFFERS, LRUS, LRU_MAX_DIRTY a LRU_MIN_DIRTY před verzí 10.0, jsou nyní určovány pomocí konfiguračního parametru BUFFERPOOL.
- CLEANERS
- RA_PAGES
- RA_THRESHOLD
- RTO_SERVER_RESTART

"Parametry RA_PAGES a RA_THRESHOLD" na stránce 5-25 popisuje parametry RA_PAGES a RA_THRESHOLD.

CLEANERS

Konfigurační parametr CLEANERS označuje počet jednotkových procesů vyčištění stránky na spuštění. Pro instalace podporující méně než 20 disků se pro každý disk obsahující databázová data doporučuje jeden jednotkový proces vyčištění stránky. Pro instalace podporující 20 až 100 disků se pro každé dva disky doporučuje jeden jednotkový proces vyčištění stránky. Pro ještě větší instalace se doporučuje jeden jednotkový proces vyčištění stránky pro každé čtyři disky. Pokud zvětšíte počet front LRU, musíte úměrně tomu zvětšit také počet jednotkových procesů vyčištění stránky.

BUFFERPOOL

Konfigurační parametr BUFFERPOOL určuje počet nejdéle nepoužívaných (LRU) dotazů, který se má nastavit ve společné oblasti vyrovnávací paměti sdílené paměti. Společná oblast vyrovnávací paměti je rozdělena mezi dotazy LRU. Konfigurování více dotazů LRU umožňuje, aby mohlo pracovat více čističů stránek a snižuje velikost každého dotazu LRU. U systému s jedním procesorem byste měli nastavit volbu **lrus** konfiguračního parametru BUFFERPOOL minimálně na hodnotu 4. U systému s více procesory nastavte volbu lrus minimálně na hodnotu 4 nebo NUMCPUVPS, která je větší.

Hodnoty **lrus**, **lru_max_dirty** a**lru_min_dirty** řídí, jak často jsou mezi kontrolními body stránky vyprázdněny na disk. Automatické ladění LRU ovlivňuje všechny společné oblasti vyrovnávací paměti a upravuje hodnoty **lru_min_dirty** a **lru_max_dirty** v konfiguračním parametru BUFFERPOOL.

Pokud zvýšíte hodnoty **lru_max_dirty** a **lru_min_dirty**, abyste zlepšili propustnost transakce, neměňte rozestup mezi **lru_max_dirty** a **lru_min_dirty**.

Pokud je společná oblast vyrovnávací paměti velmi velká a pokud se během zpracování kontrolního bodu vyskytuje blokování transakce, podívejte se do protokolu zprávy, abyste určili zdroj spouštění blokování transakce. Pokud je velikost fyzického nebo logického protokolu kriticky nízká a spouští blokování transakce, zvětšete velikost zdroje způsobujícího blokování transakce. V případě, že nemůžete velikost zdroje zvětšit, uvažte možnost zvýšení agresivity vyprazdňování LRU tím, že snížíte nastavení hodnot **lru_min_dirty** a **lru_max_dirty**, aby měl server během zpracování kontrolních bodů méně stránek k vyprázdnění na disk.

Chcete-li sledovat procento neaktualizovaných stránek v dotazech LRU, použijte příkaz **onstat -R**. Pokud počet neaktualizovaných stránek neustále převyšuje limit **lru_max_dirty**, máte příliš málo dotazů LRU nebo čističů stránek. Nejprve použijte konfigurační parametr BUFFERPOOL a zvětšete počet dotazů LRU. Pokud bude procento neaktualizovaných stránek stále převyšovat limit **lru_max_dirty**, použijte parametr CLEANERS a zvětšete počet čističů stránek.

RTO_SERVER_RESTART

Konfigurační parametr RTO_SERVER_RESTART umožňuje použít standardy cílové doby obnovy (RTO) pro nastavení množství času v sekundách, během kterého se dynamický server musí obnovit po problému poté, co dynamický server restarujete a uvedete do režimu online nebo do klidového režimu. Pokud je tento konfigurační parametr povolený, databázový server:

- Automaticky upraví počet virtuálních procesorů AIO a jednotkových procesů vyčištění.
- Automatický vyladí vyprazdňování LRU.

  Konfigurační parametr AUTO_LRU_TUNING určuje, zda je při spuštění serveru automatické ladění LRU zapnuté nebo vypnuté.

Další informace o konfiguračním parametru RTO_SERVER_RESTART naleznete v *Příručka administrátora serveru IBM Informix Dynamic Server* nebo v *IBM Informix Dynamic Server Administrator's Reference*.

Konfigurační parametry, které ovlivňují zálohování a obnovení

Zálohování a obnovení ovlivňují následující konfigurační parametry:

- BAR_IDLE_TIMEOUT
- BAR_MAX_BACKUP
- BAR_NB_XPORT_COUNT
- BAR_PROGRESS_FREQ
- BAR_XFER_BUF_SIZE

Jen pro UNIX

- LTAPEBLK
- LTAPEDEV
- LTAPESIZE
- TAPEBLK
- TAPEDEV
- TAPESIZE

Konec Jen pro UNIX

Konfigurační parametry obslužného programu ON-Bar

Konfigurační parametr BAR_IDLE_TIMEOUT určuje maximální počet minut, během nichž je proces onbar-worker nečinný před svým ukončením.

Konfigurační parametr BAR_MAX_BACKUP určuje maximální počet procesů zálohování na jeden příkaz obslužného programu ON–Bar. Tento konfigurační parametr také umožňuje definovat stupeň podobnosti určující, kolik procesů spustit, aby běžely souběžně, včetně procesů pro zálohování a obnovu celého systému. Když je dosaženo daného počtu spuštěných procesů, další procesy se spustí jen tehdy, když spuštěný proces dokončí svou činnost.

Parametr BAR_NB_XPORT_COUNT určuje počet vyrovnávacích pamětí pro data sdílené paměti pro každý proces zálohování nebo obnovení.

Parametr BAR_PROGRESS_FREQ určuje v minutách, jak často se v protokolu aktivit zobrazují zprávy o průběhu zálohování a obnovení.

Parametr BAR_XFER_BUF_SIZE určuje velikost vyrovnávacích pamětí v počtu stránek.

Další informace o těchto konfiguračních parametrech naleznete v příručce *IBM Informix Backup and Restore Guide*.

Konfigurační parametry ontape (systém UNIX)

Parametry LTAPEBLK, LTAPEDEV a TAPESIZE určují velikost bloku, zařízení a velikost pásky pro zálohy logického protokolu provedené pomocí obslužného programu **ontape**. Konfigurační parametr TAPEBLK určuje velikost bloku pro zálohy databáze provedené pomocí obslužných programů **ontape**, **onload** a **onunload**. Parametr TAPEDEV určuje páskové zařízení. Parametr TAPESIZE určuje velikost pásky pro tyto zálohy. Informace o těchto obslužných programech a o specifických doporučeních pro operace zálohování a obnovení naleznete v příručce *IBM Informix Backup and Restore Guide*.

Konfigurační parametry, které ovlivňují odvolání a obnovu

Rychlou obnovu ovlivňují následující konfigurační parametry:

- OFF_RECVRY_THREADS
- ON_RECVRY_THREADS
- PLOG_OVERFLOW_PATH
- RTO_SERVER_RESTART

OFF_RECVRY_THREADS a ON_RECVRY_THREADS

Konfigurační parametry OFF_RECVRY_THREADS a ON_RECVRY_THREADS v tomto pořadí určují počet jednotkových procesů obnovy, které pracují, když databázový server provádí studené, teplé nebo rychlé obnovení. Nastavení parametru OFF_RECVRY_THREADS řídí studená obnovení a nastavení parametru ON_RECVRY_THREADS řídí rychlou obnovu a teplá obnovení.

Chcete-li zlepšit výkon rychlé obnovy, zvětšete počet jednotkových procesů rychlé obnovy pomocí konfiguračního parametru ON_RECVRY_THREADS. Počet jednotkových procesů by měl obvykle odpovídat počtu tabulek nebo fragmentů, které jsou často aktualizované pro přehrání transakcí zaznamenaných v logickém protokolu.

Dalším odhadem je počet tabulek nebo fragmentů, u kterých probíhají časté aktualizace. U hostitele s jedním procesorem by počet jednotkových procesů neměl být menší než 10 a větší než 30 nebo 40. V určitém okamžiku převáží zahlcení, které je spojené s každým jednotkovým procesem, nad výhodami paralelních jednotkových procesů.

Teplé obnovení se odehrává souběžně s ostatními databázovými operacemi. Chcete-li snížit vliv teplého obnovení na ostatní uživatele, můžete mu přidělit méně jednotkových procesů, než byste přidělili studenému obnovení. Chcete-li však znovu současně spustit transakce logického protokolu během studeného obnovení, zadejte v parametru ON_RECVRY_THREADS více jednotkových procesů.

PLOG_OVERFLOW_PATH

Konfigurační parametr PLOG_OVERFLOW_PATH určuje umístění souboru disku (pojmenovaného **plog_extend.servernum**), který databázový server používá v případě přetečení soubory fyzického protokolu během rychlé obnovy.

Databázový server soubor **plog_extend.servernum** odstraní, když je během rychlé obnovy proveden první kontrolní bod.

RTO_SERVER_RESTART

Konfigurační parametr RTO_SERVER_RESTART umožňuje použít standardy cílové doby obnovy (RTO) pro nastavení množství času v sekundách, během kterého se dynamický server musí obnovit po problému poté, co dynamický server restarujete a uvedete do režimu online nebo do klidového režimu.

Další informace o těchto parametrech naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference* a v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Konfigurační parametry, které ovlivňují replikaci dat a auditování

Replikace dat a auditování jsou volitelné. Chcete-li dosáhnout okamžitého zlepšení výkonu, můžete tyto funkce zakázat za předpokladu, že vám to umožní operační požadavky na váš systém.

Replikace dat

Replikace dat obvykle přidá do vstupu - výstupu logického protokolu zahlcení úměrné aktivitě protokolování. Konfigurace optimalizující aktivitu protokolování bez replikace dat je vhodná také pro optimalizaci protokolování s replikací dat.

Výkon replikace dat ovlivňují následující konfigurační parametry:

- DRINTERVAL
- DRTIMEOUT

Konfigurační parametr DRINTERVAL označuje, jestli je vyrovnávací paměť replikace dat vyprázdněna do sekundárního databázového serveru synchronně nebo asynchronně. Pokud je tento parametr nastavený, aby vyprazdňoval asynchronně, určuje interval mezi vyprázděními. Každé vyprázdnění ovlivňuje CPU a odesílá skrz síť data do sekundárního databázového serveru.

Konfigurační parametr DRTIMEOUT určuje interval, po který čeká jeden databázový server na potvrzení přenosu od druhého serveru. V případě, že primární databázový server neobdrží očekávané potvrzení, přidá informaci o transakci do souboru pojmenovaném v konfiguračním parametru DRLOSTFOUND. Pokud sekundární databázový server neobdrží žádné potvrzení, změní režim replikace dat tak, jak určuje konfigurační parametr DRAUTO. Další informace o replikaci dat naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Auditování

Účinek auditování na výkon je do značné míry určen tím, jaké událostí auditování vyberete k zaznamenání. V závislosti na tom, kteří uživatelé a události jsou auditované, vliv na výkon se může hodně lišit. Události, které se nevyskytují často, jako na příklad požadavky na připojení do databáze, mají malý vliv na výkon. Časté události, jako například požadavky na čtení libovolné řádky, můžou generovat velké množství aktivity auditování. Čím větší je množství uživatelů, pro které jsou takovéto události auditované, tím větší to bude mít vliv na výkon. Další informace o auditování naleznete v příručce *IBM Informix Security Guide*.

Výkon auditování ovlivňují následující konfigurační parametry:

- ADTERR
- ADTMODE

Konfigurační parametr ADTERR určuje, jestli má databázový server zastavit zpracování uživatelské relace, pro kterou záznam auditu zjistí chybu. Když je parametr ADTERR nastavený, aby takovou relaci zastavil, bude se doba odezvy této relace pravděpodobně znehodnocovat, dokud neuspěje jeden z po sobě následujících pokusů o zapsání záznamu auditu.

Konfigurační parametr ADTMODE povoluje nebo zakazuje auditování na základě záznamů auditu, které určíte pomocí obslužného programu **onaudit**. Záznamy jsou zapsány do souborů v adresáři, který určuje parametr AUDITPATH. Parametr AUDITSIZE určuje velikost každého souboru záznamu auditu.

Ladění LRU

Nastavení LRU pro vyprázdnění každé vyrovnávací paměti mezi kontrolními body nejsou kritická pro výkon kontrolního bodu. Nastavení LRU jsou nezbytná pouze pro udržení dostatečného množství čistých stránek pro náhradu stránky.

Výchozí nastavení pro vyprázdnění LRU je 50 procent pro **lru_min_dirty** a 60 procent pro **lru_max_dirty**.

Pokud byl databázový server konfigurován pro agresivnější vyprázdnění LRU kvůli výkonu kontrolního bodu, můžete snížit vyprázdnění LRU minimálně na výchozí hodnoty.

Databázový server automaticky vyladí vyprázdnění LRU, kdykoliv se vyskytne náhrada stránky. Poté, co se vyskytnul kontrolní bod a pokud se v průběhu předešlého intervalu kontrolního bodu vyskytnul zápis nahrazení stránky na popředí, databázový server zvýší nastavení LRU o 10 procent a bude pokračovat ve zvyšování vyprázdění LRU v každém následujícím kontrolním bodu, dokud se nezastaví zápisy nahrazení stránky na popředí, nebo pokud neklesne hodnota **lru_max_dirty** dané společné oblasti vyrovnávací paměti pod 10 procent. Například, pokud se vyskytne zápis nahrazení stránky na popředí a nastavení LRU pro společnou oblast vyrovnávací paměti jsou 80 a 90, databázový server upraví tyto hodnoty na 76 a 85,5.

Dodatek k zápisům na popředí - vyprázdnění LRU je laděno agresivněji, kdykoliv chyba stránky nahradí vyrovnávací paměti s vysokou prioritou a vyrovnávací paměti s nevelkou prioritou jsou v modifikovaném dotazu LRU. Automatické úpravy LRU udělají vyprázdnění LRU pouze agresivnější, nesníží vyprázdnění LRU. Automatické úpravy LRU nejsou trvalé a nejsou zaznamenány v souboru ONCONFIG.

Vyprázdnění LRU je resetováno na hodnoty obsažené v souboru ONCONFIG, ze kterého se spustí databázový server.

Konfigurační parametr AUTO_LRU_TUNING určuje, zda je při spuštění serveru automatické ladění LRU zapnuté nebo vypnuté.

Kapitola 6. Úvahy o výkonu tabulek

Obsah kapitoly

Tato kapitola popisuje úvahy o výkonu týkající se nefragmentovaných tabulek a tabulkových fragmentů. Vztahuje se k následujícím problémům:

- Umístění tabulek na disku ke zvýšení propustnosti a zmenšení soupeření
- Odhady místa pro tabulky a prostory blobspace
- Správa prostoru sbspace
- Správa oblastí
- Změny tabulek, které přidávají nebo odstraňují historická data
- Denormalizace databáze ke snížení zahlcení

Umístění tabulek na disk

Tabulky podporované databázovým serverem jsou umístěny na jedné nebo více částech disků. Umístění tabulky na disku lze řídit při vytvoření tabulky přiřazením do prostoru dbspace. Prostor dbspace se skládá z jednoho nebo více bloků. Každý blok odpovídá celému nebo části oddílu disku. Pokud přiřadíte bloky prostorům dbspace, zpřístupníte diskový prostor v těchto blocích pro ukládání tabulek nebo fragmentů tabulek.

Pokud nakonfigurujete bloky a přidělíte je prostorům dbspace, musíte vztáhnout velikost prostorů dbspace k tabulkám nebo fragmentům, které mají jednotlivé prostory dbspace obsahovat. Velikost tabulky můžete odhadnout podle pokynů uvedených v části “Odhad velikosti tabulky” na stránce 6-6.

Administrátor databáze (DBA), který je odpovědný za vytvoření tabulky, přiřazuje tabulku do prostoru dbspace jedním z následujících způsobů:

- Pomocí klauzule IN DBSPACE příkazu CREATE TABLE
- Pomocí prostoru dbspace aktuální databáze

  Nejnovější příkaz DATABASE nebo CONNECT vyvolaný uživatelem DBA před vyvoláním příkazu CREATE TABLE nastavuje aktivní databázi.

Uživatel DBA může fragmentovat tabulku mezi více prostorů dbspace, jak je popsáno v příručce “Naplánování strategie fragmentace” na stránce 9-2, nebo přesunout tabulku do jiného prostoru dbspace pomocí příkazu ALTER FRAGMENT. Příkaz ALTER FRAGMENT poskytuje nejjednodušší způsob úpravy umístění tabulek. Ale zatímco databázový server provádí úpravu tabulky, není tabulka k dispozici. Přesun tabulky nebo fragmentu naplánujte

na čas, ve kterém bude mít vliv na nejmenší počet uživatelů. Popis příkazu ALTER FRAGMENT naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Pro přesun tabulek mezi prostory dbspace existují další metody. Uživatel DBA může uvolnit data z tabulky a přesunout data do jiného prostoru dbspace pomocí příkazů jazyka SQL LOAD a UNLOAD, jak je popsáno v příručce *IBM Informix Guide to SQL: Syntax*. Administrátor databázového serveru může provádět stejné akce pomocí nástrojů **onload** a **onunload**, jak je popsáno v příručce *IBM Informix Migration Guide*, nebo pomocí High-Performance Loader (zavaděč HPL), jak je popsáno v příručce *IBM Informix High-Performance Loader User's Guide*.

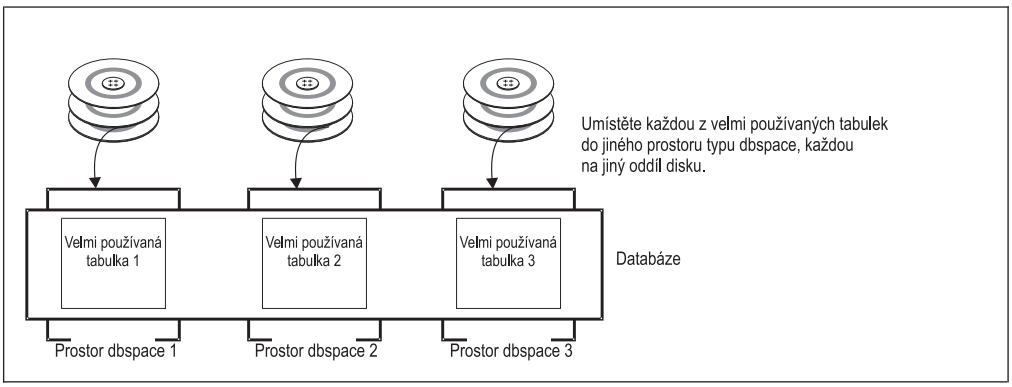
Přesun tabulek mezi databázemi pomocí příkazů LOAD a UNLOAD, nástrojů **onload** a **onunload** nebo HPL má vliv na interval, po kterou se tabulka kopíruje na pásku a znovu zavádí do systému. Tyto intervaly představují okna zranitelnosti, během kterých může vzniknout nekonzistence tabulky se zbytkem databáze. Chcete-li zajistit, aby nedošlo ke vzniku nekonzistence tabulky, musíte omezit přístup k verzi, která je uložena na disku během přenosu.

V závislosti na velikosti, strategii fragmentace a indexech přidružených k tabulce může být rychlejší uvolnit tabulku a znovu ji zavést, než upravit fragmentaci. U jiných tabulek může být rychlejší upravit fragmentaci. Jaká metoda je rychlejší pro tabulku, kterou chcete přesunout nebo znovu rozdělit, můžete zjistit experimentálně.

Izolace vysoce používaných tabulek

Tabulku s vysokou aktivitou vstupu - výstupu můžete umístit na vyhrazené diskové zařízení a snížit tak soupeření o data uložená v této tabulce. Pokud mají disky jiné úrovně výkonu, můžete více používané tabulky umístit na rychlejší jednotky. Pokud umístíte dvě vysoce používané tabulky na oddělená disková zařízení, snížíte tím soupeření o přístup na disk, pokud u těchto dvou tabulek dochází k současnému vstupně - výstupnímu přístupu z několika aplikací nebo dochází ke spojení tabulek.

Chcete-li izolovat vysoce používané tabulky na vlastní diskové zařízení, přiřaďte zařízení do bloku, tento blok do prostoru dbspace a umístěte tabulku do tohoto prostoru dbspace. Obrázek 6-1 znázorňuje tři vysoce používané tabulky, každou v odděleném prostoru dbspace, které jsou umístěny na třech discích.

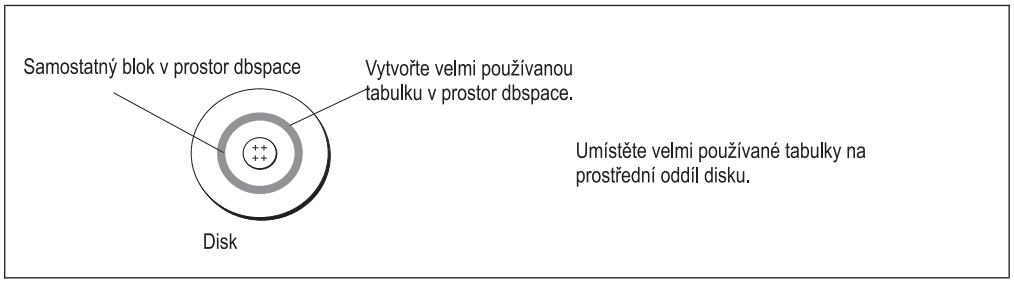


Obrázek 6-1. Izolace vysoce používaných tabulek

Umístění vysoce používaných tabulek na střední oddíly disků

Chcete-li snížit pohyb diskových hlav, umístěte nejčastěji používaná data na oddíly blízko střednímu pásu disku (ne blízko středu disku nebo kraje disku), jak zobrazuje Obrázek 6-2. Tento přístup minimalizuje pohyby diskových hlav při přístupu k datům v často

požadovaných tabulkách.



Obrázek 6-2. Disk s vysoce používanou tabulkou umístěnou ve středních oddílech

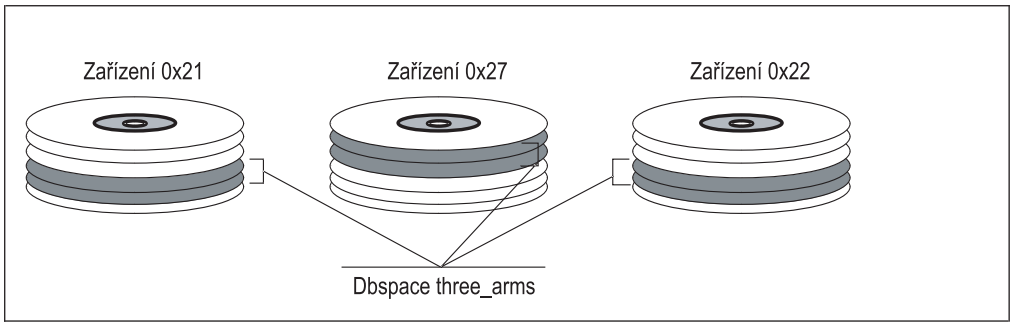
Chcete-li umístit vysoce používané tabulky na střední oddíl disku, vytvořte přímé zařízení, které se skládá z cylindrů ve střední části mezi středem a vnějším krajem disku. (Postup vytvoření přímého zařízení naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server* pro příslušný operační systém.) Přidělte blok, aby byl přidružen k přímému zařízení, jak popisuje část *IBM Informix Dynamic Server Administrator's Reference*. Pak vytvořte prostor dbspace se tímto stejným blokem jako počáteční a jediný blok. Až vytvoříte vysoce používanou tabulku, umístěte ji do tohoto prostoru dbspace.

Použití více disků

Tato část pojednává o použití několika disků pro prostory dbspace, logické protokoly a dočasné prostory dbspace.

Použití více disků v prostoru dbspace

Prostor dbspace může obsahovat více bloků a každý blok může představovat jiný disk. Maximální velikost bloku je 4 TB. Toto uspořádání umožňuje rozdělovat data do prostorů dbspace mezi více disků. Obrázek 6-3 znázorňuje prostor dbspace rozdělený na tři disky.



Obrázek 6-3. Prostor dbspace rozdělený na tři disky

Použití více disků v prostoru dbspace pomáhá distribuovat vstup - výstup mezi prostory dbspace, které obsahují několik malých tabulek. Protože tento typ distribuovaného prostoru dbspace nelze použít v paralelních databázových dotazech (PDQ), je doporučeno pomocí technik fragmentace tabulek popsaných v části “Navržení schématu distribuce” na stránce 9-6 rozdělit velké, vysoce používané tabulky mezi více prostorů dbspace.

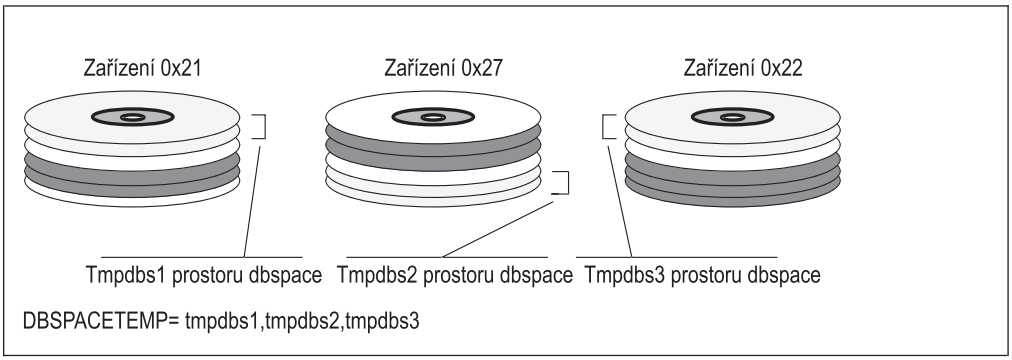
Použití více disků v pro logické protokoly

Chcete-li zlepšit výkon logického zálohování, můžete distribuovat logické protokoly v různých prostorech dbspace na více disků metodou cyklické obsluhy. Toto schéma umožňuje databázovému serveru zálohovat protokoly na jeden disk a současně protokolovat na ostatní disky.

Chcete-li zvýšit výkon snížením soupeření vstupu - výstupu na jedno zařízení, uchovávejte logické protokoly a fyzické protokoly na oddělených zařízeních. Logické a fyzické protokoly se vytváří v prostoru dbspace při inicializaci databázového serveru. Po inicializaci je můžete přesunout do jiných prostorů dbspace.

Rozložení dočasných tabulek a souborů řazení mezi několik disků

K definování několika prostorů dbspace pro dočasné tabulky a soubory řazení použijte příkaz **onspaces -t**. Pokud tyto prostory dbspace umístíte na různé disky a uvedete je v konfiguračním parametru DBSPACETEMP, budete moci operace vstupu - výstupu spojené s použitím dočasných tabulek a souborů řazení rozložit na více disků, jak ukazuje Obrázek 6-4. Prostory dbspace, které obsahují běžné tabulky, můžete uvést v parametru DBSPACETEMP.



Obrázek 6-4. Prostory dbspace pro dočasné tabulky a soubory řazení

Uživatelé mohou určit vlastní seznamy prostorů dbspace pro dočasné tabulky a soubory řazení pomocí proměnné prostředí **DBSPACETEMP**. Podrobnosti naleznete v části “Konfigurace prostorů dbspace pro dočasné tabulky a soubory řazení” na stránce 5-7.

Úvahy o zálohování a obnovení

Pokud se rozhodujete, kam umístit tabulky a fragmenty, vezměte v úvahu, že v případě selhání zařízení obsahujícího prostor dbspace budou všechny tabulky v tomto prostoru dbspace nepřístupné. Potřeba omezit nedostupnost dat v případě poruchy disku může mít vliv na tabulky, které v konkrétním prostoru dbspace budete chtít seskupit.

V případě selhání kritického prostoru dbspace musíte provést studené obnovení, ale v případě selhání nekritického prostoru dbspace musíte provést pouze teplé obnovení. Potřeba minimalizovat dopad studených obnovení může mít vliv na prostor dbspace, který používáte k ukládání kritických dat. Další informace naleznete v příručkách *IBM Informix Backup and Restore Guide* a *IBM Informix Backup and Restore Guide*.

Zlepšení výkonu nefragmentovaných tabulek a fragmentů tabulek

Na určitou tabulku nebo fragment tabulky mají vliv následující faktory:

- Umístění tabulky nebo fragmentu, jak je popsáno v předchozí části
- Velikost tabulky nebo fragmentu
- Použitá strategie indexování
- Velikost a umístění oblastí tabulky s ohledem na ostatní
- Frekvence přístupu k tabulce

Odhad velikosti tabulky

Tato část popisuje metody výpočtu přibližných velikostí tabulek (ve stránkách disku).

Popis výpočtů velikosti v případě indexů naleznete v části “Odhad indexových stránek” na stránce 7-1.

Stránky disku přidělené do tabulky se společně označují jako prostor *tblspace*. Prostor tblspace obsahuje datové stránky. Samostatné prostory obsahují indexové stránky. Pokud jsou jednoduché velké objekty (data typu TEXT nebo BYTE) přidružené k tabulce, která není uložena v alternativním prostoru dbspace, jsou v prostoru tblspace zahrnuty také stránky, které obsahují jednoduché velké objekty.

Prostor tblspace se nevztahuje k žádné pevné oblasti v prostoru dbspace. Datové oblasti a indexy, ze kterých se skládá tabulka, mohou být rozptýleny v prostoru dbspace.

Velikost tabulky zahrnuje všechny stránky v prostoru tblspace: datové stránky a stránky, ve kterých jsou uloženy jednoduché velké objekty. Stránky blob, které jsou uloženy v odděleném prostoru blobspace nebo na optickém podsystému, nejsou zahrnuty v prostoru tblspace a nepočítají se jako část velikosti tabulky. Následující části popisují způsob, jak odhadnout počet stránek jednotlivých typů stránek v prostoru tblspace.

Tip: Pokud již existuje odpovídající vzorová tabulka nebo můžete vytvořit vzorovou tabulku realistické velikosti se simulovanými daty, nemusíte vytvářet odhady. Přesné hodnoty můžete získat spuštěním obslužného programu **oncheck -pt**.

Odhad datových stránek

Způsob odhadu datových stránek závisí na tom, zda tabulka obsahuje řádky fixní nebo proměnné velikosti.

Odhad tabulek s řádky fixní velikosti

Chcete-li odhadnout velikost tabulky (ve stránkách) s řádky fixní délky, proveďte následující kroky. Tabulka s řádky fixní délky nemá sloupce typu VARCHAR ani NVARCHAR.

Postup odhadnutí velikosti stránky, velikosti řádku, počtu řádků a počtu datových stránek:

1. Velikost stránky získáte pomocí příkazu **onstat -b** .

   Velikost stránky zobrazuje pole **velikost vyrovnávací paměti** na posledním řádku tohoto výstupu.
2. Odečtením čísla 28 od této hodnoty započtěte hlavičku, která se zobrazuje na každé datové stránce.

   Výsledná hodnota se označuje jako *pageuse*.
3. Chcete-li vypočítat velikost řádku, sečtěte šířky všech sloupců v definici tabulky. Sloupce TEXT a BYTE používají po 56 bajtech.

   Pokud jste již tabulku vytvořili, můžete získat velikost řádku pomocí následujícího příkazu jazyka SQL:

   ```
   SELECT rowsize FROM systables WHERE tabname =
      'název-tabulky';
   ```
4. Odhadněte předpokládaný počet řádků, které bude tabulka obsahovat.

   Tento počet je označen názvem *rows*.

   Postup výpočtu počtu datových stránek, které potřebuje tabulka, se liší v závislosti na tom, zda je velikost řádku menší nebo větší než hodnota *pageuse*.

5. Pokud je velikost řádku menší nebo rovna než hodnota *pageuse*, použijte k výpočtu počtu datových stránek následující vzorec.

   Notace funkce **trunc()** označuje, že se zaokrouhluje dolů na nejbližší celé číslo.

   ```
   data_pages = rows / trunc(pageuse/(rowsize + 4))
   ```

   Maximální počet řádek na stránce je 255, bez ohledu na velikost řádku.

   Důležité: I když je maximální velikost řádku, který databázový server přijímá, přibližně 32 kB, snižuje se výkon při překročení velikosti stránky. Informace o zvýšení výkonu pomocí rozdělení širokých tabulek naleznete v části “Zlepšení výkonu pomocí denormalizace datového modelu” na stránce 6-38.

6. Pokud je velikost řádku větší, než hodnota *pageuse*, rozdělí databázový server řádek mezi stránky.

   Stránka, která obsahuje počáteční část řádku, se nazývá *domovská stránka*. Stránky, které obsahují následující části řádku, se nazývají *zbývající stránky*. Pokud se řádek nachází na více než dvou stránkách, jsou některé zbývající stránky zcela vyplněné daty tohoto řádku. Pokud koncová část řádku používá méně než jednu stránku, lze ji zkombinovat s koncovými částmi jiných řádků, aby se vyplnily dílčí zbývající stránky. Počet datových stránek je součtem domovských stránek, úplných zbývajících stránek a dílčích zbývajících stránek.

   a. Vypočtěte počet domovských stránek.

      Počet domovských stránek je stejný jako počet řádků:

      ```
      homepages = rows
      ```

   b. Vypočtěte počet zbývajících úplných stránek.

      Nejdříve vypočtěte podle následujícího vzorce velikost zbytku řádku:

      ```
      remsize = rowsize - (pageuse + 8)
      ```

      Pokud je hodnota *remsize* menší, než hodnota *pageuse* - 4, neexistují žádné úplné zbývající stránky.

      Pokud je hodnota *remsize* větší, než hodnota *pageuse* - 4, použijte k získání počtu úplných zbývajících stránek v následujícím vzorci hodnotu *remsize*:

      ```
      fullrempages =  rows * trunc(remsize/(pageuse - 8))
      ```

   c. Vypočtěte počet dílčích zbývajících stránek.

      Nejdříve vypočtěte velikost zbytku dílčího řádku, který zbyl po započítání domovských a úplných zbývajících stránek určitého řádku. V následujícím vzorci funkce **remainder()** označuje, že musíte po dělení vzít v úvahu zbytek:

      ```
      partremsize = remainder(rowsize/(pageuse - 8)) + 4
      ```

      Databázový server používá ke zjištění počtu dílčích zbývajících stránek určité prahy velikosti vzhledem k velikosti stránky. Pomocí následujícího vzorce vypočítejte poměr dílčího zbytku ke stránce:

      ```
      partratio = partremsize/pageuse
      ```

      Vypočtěte počet dílčích zbývajících stránek podle odpovídajícího vzorce v následující tabulce.

Hodnota partratio	**Vzorec k výpočtu počtu dílčích zbývajících stránek**
Méně než 0,1	partrempages = *rows*/(trunc((*pageuse*/10)/*remsize*) + 1)
Méně než 0,33	partrempages = *rows*/(trunc((*pageuse*/3)/*remsize*) + 1)
0,33 nebo více	partrempages = *rows*

   d. Podle následujícího vzorce sečtěte celkový počet stránek:

```
tablesize = homepages + fullrempages + partrempages
```

Odhad tabulek s řádky proměnné velikosti

Pokud tabulka obsahuje jeden nebo více sloupců typu VARCHAR nebo NVARCHAR, mohou mít její řádky proměnnou délku. Tyto proměnné délky představují nejistotu při výpočtech. Musíte vytvořit odhad charakteristické velikosti jednotlivých sloupců VARCHAR založený na znalosti dat a použít tuto hodnotu při vytváření odhadů.

Důležité: Když databázový server přiděluje prostor řádkům s proměnnou velikostí a neexistuje prostor pro další řádek maximální velikosti, považuje se stránka za plnou.

Chcete-li odhadnout velikost tabulky s proměnnou délkou řádků, musíte vytvořit následující odhady a podle informací o datech vybrat hodnotu mezi nimi:

- Maximální velikost tabulky, kterou vypočítáte podle maximální možné šířky všech sloupců VARCHAR a NVARCHAR
- Plánovaná velikost tabulky, kterou vypočítáte podle charakteristické šířky jednotlivých sloupců VARCHAR a NVARCHAR

Odhad maximálního počtu datových stránek:

1. Chcete-li vypočítat hodnotu *rowsize*, sečtěte maximální hodnoty všech šířek sloupců.
2. Tuto hodnotu použijte pro hodnotu *rowsize* a proveďte výpočty popsané v části “Odhad tabulek s řádky fixní velikosti” na stránce 6-6. Výsledná hodnota se označuje jako *maxsize*.

Odhad plánovaného počtu datových stránek:

1. Chcete-li vypočítat hodnotu *rowsize*, sečtěte charakteristické hodnoty jednotlivých sloupců proměnné šířky. Jako charakteristickou šířku sloupce je doporučeno použít nejčastěji se vyskytující šířku tohoto sloupce. Pokud nemáte přístup k datům nebo nechcete šířky tabelovat, můžete vybrat určité zlomky maximální šířky, například 2/3 (0,67).
2. Tuto hodnotu použijte pro hodnotu *rowsize* a proveďte výpočty popsané v části “Odhad tabulek s řádky fixní velikosti” na stránce 6-6. Výsledná hodnota se označuje jako *projsize*.

Výběr střední hodnoty velikosti tabulky

Skutečná velikost tabulky může klesnout mezi hodnotu *projsize* a *maxsize*. Podle znalosti dat zvolte v tomto rozsahu hodnotu, která se zdá nejvhodnější. Čím méně informací o datech máte, tím by měl být odhad opatrnější (vyšší).

Odhad stránek, které zabírají jednoduché velké objekty

Stránky blobpage mohou být uloženy v prostoru dbspace, ve kterém je uložena tabulka, nebo v prostoru blobspace. Další informace o tom, kdy používat prostor blobspace, naleznete v části “Ukládání jednoduchých velkých objektů v odděleném prostoru blobspace” na stránce 6-9.

Následující metody odhadu stránek blobpage vytvářejí opatrný (vysoký) odhad, protože jednotlivé sloupce typu TEXT a BYTE nemusí nezbytně obsadit celou stránku blobpage v prostoru tblspace. Jinými slovy, stránka blobpage v prostoru tblspace může obsahovat více sloupců TEXT a BYTE.

Odhad počtu stránek blobpage:

1. Získejte velikost stránky příkazem **onstat -b**.
2. Vypočtěte použitelnou část stránky blobpage pomocí následujícího vzorce:

```
bpuse = pagesize - 32
```

3. Pro každý bajt velikosti blobsize *n* vypočtěte pomocí následujícího vzorce počet stránek, které bajt zabírá (*bpages_n*):

   ```
   bpages1 = ceiling(bytesize1/bpuse)
   bpages2 = ceiling(bytesize2/bpuse)
   ...
   bpages_n = ceiling(bytesize_n/bpuse)
   ```

   Funkce **ceiling()** označuje zaokrouhlení na nejbližší vyšší celočíselnou hodnotu.

4. Následujícím způsobem sečtěte celkový počet stránek všech jednoduchých velkých objektů:

   ```
   blobpages = bpages1 + bpages2 + ... + bpagesn
   ```

Nebo můžete odhad založit na střední hodnotě velikosti jednoduchých velkých objektů (data typu TEXT nebo BYTE), to znamená velikost dat jednoduchých velkých objektů, které se vyskytují nejčastěji. Tento způsob je méně přesný, ale je snadnější jej vypočítat.

Odhad počtu stránek blobpage založený na střední hodnotě velikosti jednoduchých velkých objektů:

1. Následujícím způsobem vypočtěte počet stránek požadovaných pro jednoduché velké objekty střední velikosti:

   ```
   mpages = ceiling(mblobsize/bpuse)
   ```

2. Tuto hodnotu následujícím způsobem vynásobte celkovým počtem jednoduchých velkých objektů:

   ```
   blobpages = blobcount * mpages
   ```

Ukládání jednoduchých velkých objektů v odděleném prostoru blobspace

Pokud vytvoříte na magnetickém disku sloupec jednoduchého velkého objektu, můžete ukládat data sloupce v prostoru tblspace nebo v odděleném prostoru blobspace. Výkon můžete obvykle zlepšit ukládáním dat jednoduchých velkých objektů v odděleném prostoru blobspace, jak je popsáno v části “Odhad stránek, které zabírají jednoduché velké objekty” na stránce 6-8, a ukládáním inteligentních velkých objektů a uživatelsky definovaných dat v prostoru sbspace.

(Jednoduché velké objekty můžete také uložit na optické médium, ale tato část se nevztahuje na ukládání jednoduchých velkých objektů tímto způsobem.)

V následujícím příkladu je hodnota typu TEXT uložena v prostoru tblspace a hodnota typu BYTE je uložena v prostoru blobspace s názvem **rasters**:

```
CREATE TABLE examptab
   (
   pic_id SERIAL,
   pic_desc TEXT IN TABLE,
   pic_raster BYTE IN rasters
   )
```

Hodnota typu TEXT nebo BYTE je vždy uložena odděleně od řádků tabulky, u řádků je uložen pouze 56bajtový deskriptor. Jednoduchý velký objekt ale zabírá nejméně jednu diskovou stránku. Jednoduchý velký objekt, na který deskriptor ukazuje, může být uložen ve stejné sadě oblastí na disku jako řádky tabulky (ve stejném prostoru tblspace) nebo v odděleném prostoru blobspace.

Pokud jsou jednoduché velké objekty uloženy v prostoru tblspace, jsou příslušné datové stránky promíchány se stránkami, které obsahují řádky, a mohou významně zvýšit velikost tabulky. Pokud databázový server čte pouze řádky a nečte jednoduché velké objekty,

pohybuje se disková hlava dále, než pokud jsou stránky blobpage uloženy odděleně. V následujících situacích prohledává databázový server pouze stránky řádků:

- Když provádí operaci SELECT, nezískává sloupec jednoduchého velkého objektu
- Když používá výraz filtru k testování řádků

Další úvaha je o tom, že se diskový vstup - výstup z prostoru dbspace ukládá do sdílené vyrovnávací paměti databázového serveru. Stránky se ukládají v případě, že budou brzy znovu třeba, a po zapsání stránek může žádající program pokračovat, než je skutečný zápis na disk vykonán. Protože se předpokládá, že data prostoru blobspace zabírají velký prostor, neukládá se diskový vstup - výstup z prostorů blobspace do vyrovnávací paměti a žádající program nemůže pokračovat, dokud se celý výstup nezapíše do prostoru blobspace.

Lepšího výkonu lze dosáhnout uložením sloupce jednoduchého velkého objektu do prostoru blobspace při následujících okolnostech:

- Když jsou jednotlivé datové položky větší než jedna nebo dvě stránky
- Když je počet stránek dat typu TEXT nebo BYTE více než polovina počtu stránek datových řádků

Odhad stránek prostorů tblspace pro jednoduché velké objekty

V odhadu prostoru požadovaného pro tabulku zahrňte stránky blobpage pro libovolné jednoduché velké objekty, které je třeba uložit v prostoru tblspace.

U tabulek, které jsou relativně malé a trvalé, můžete využít vliv vyhrazeného prostoru blobspace oddělením stránek řádků a stránek blobpage, jak je vysvětleno v následujících krocích.

Postup oddělení stránek řádků od stránek blobpage v prostoru dbspace:

1. Načtěte celou tabulku s řádky, ve kterých mají jednoduché velké objekty hodnotu null.
2. Vytvořte všechny indexy.

   Stránky řádků a stránky indexů jsou nyní vedle sebe.
3. Aktualizujte všechny řádky tak, aby se nainstalovaly jednoduché velké objekty.

   Stránky blobpage se nyní zobrazují za stránkami dat řádků a indexů v prostoru tblspace.

Správa velikosti první oblasti a dalších oblastí prostoru tblspace tblspace

Prostor tblspace **tblspace** je kolekce stránek, která popisuje umístění a strukturu všech prostorů tblspace v prostoru dbspace. Každý prostor dbspace má jeden prostor **tblspace**.

Pokud vytváříte prostor dbspace, můžete pomocí konfiguračních parametrů TBLTBLFIRST a TBLTBLNEXT určit velikost první oblasti a velikost dalších oblastí prostoru **tblspace** v kořenovém prostoru dbspace. Velikost počáteční oblasti a dalších oblastí pro prostor **tblspace** v jiných prostorech než kořenových můžete určit pomocí obslužného programu **onspaces**.

Zadáním velikosti počáteční oblasti a dalších oblastí můžete snížit počet oblastí prostoru tblspace **tblspace** a četnost situací, ve kterých je zapotřebí umístit oblasti prostoru tblspace **tblspace** do jiných než primárních bloků.

Možnost zadání velikosti první oblasti, která je větší než výchozí, poskytuje flexibilitu při správě prostoru. Pokud vytváříte oblast, můžete rezervovat prostor během vytvoření prostoru dbspace, a tím snížit riziko, že bude třeba vytvořit další oblasti v blocích, které nejsou výchozími bloky.

Při vytvoření prostoru dbspace můžete zadat pouze velikost první oblasti a velikost dalších oblastí. Po vytvoření prostoru dbspace již tyto velikosti nelze měnit. Navíc nelze určit velikosti oblastí pro dočasné prostory dbspace, sbspace, blobspace a externí prostory.

Pokud neurčíte velikost první oblasti a dalších oblastí pro prostor **tblspace**, použije dynamický server existující výchozí velikost oblasti.

Další informace o zadání velikosti první oblasti a velikosti dalších oblastí pro prostor **tblspace** naleznete v příručkách *Příručka administrátora serveru IBM Informix Dynamic Server* a *IBM Informix Administrator's Reference*.

Správa prostorů sbspace

Tato část popisuje následující témata týkající se prostorů sbspace:

- Odhad diskového prostoru
- Zlepšení vstupu - výstupu metadat
- Monitorování
- Změna paměťových charakteristik

Odhad stránek, které zabírají inteligentní velké objekty

V odhadu prostoru požadovaného pro tabulku vezměte v úvahu také velikost úložiště sbspace pro libovolné inteligentní velké objekty (například CLOB, BLOB nebo víceznačné datové typy), které jsou součástí tabulky. Prostor sbspace obsahuje oblasti uživatelských dat a oblasti metadat. Data typu CLOB a BLOB jsou uložena ve stránkách sbpage, které jsou umístěny v oblasti uživatelských dat. Oblast metadat obsahuje atributy inteligentních velkých objektů, například průměrnou velikost a údaj o tom, zda je inteligentní velký objekt protokolován. Další informace o prostorech sbspace naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Odhad velikosti prostoru sbspace a oblasti metadat

První blok prostoru sbspace musí obsahovat oblast metadat. Pokud přidáváte inteligentní velké objekty, přidá databázový server k této oblasti metadat další řídicí informace.

Pokud přidáte blok do prostoru sbspace po počátečním přidělení, můžete s prostorem metadat provést jednu z následujících akcí:

- Ve výchozím nastavení přidělit k novému bloku další oblast metadat.

  Tato akce poskytuje následující výhody:

  – Je jednodušší, protože databázový server automaticky vypočítá a přidělí novou oblast metadat k přidanému bloku podle průměrné velikosti inteligentního velkého objektu

  – Distribuuje operace vstupu - výstupu v oblasti metadat mezi více disků

- Používat existující oblast metadat.

  Pokud zadáte volbu **onspaces -U**, nepřidělí databázový server k novému bloku prostor metadat. Místo toho musí použít oblast metadat v některém z jiných bloků.

Databázový server kromě toho rezervuje 40 % uživatelské oblasti k použití v případě zaplnění oblasti metadat. Proto databázový server v případě zaplnění přiděleného prostoru metadat začne používat pro další řídicí informace tento rezervovaný prostor v uživatelské oblasti.

Výpočet velikosti oblasti metadat můžete přenechat databázovému serveru. Velikost oblasti metadat můžete ale chtít zadat explicitně, aby v prostoru sbspace nedošlo k zaplnění oblasti metadat a 40 procent rezervní oblasti. Velikost prostoru metadat k přidělení můžete určit pomocí jedné z následujících metod:

- Ve volbě **onspaces -Df** určete značku AVG_LO_SIZE.

  Databázový server tuto hodnotu používá k výpočtu velikosti oblasti metadat, která je přidělena, pokud není zadána volba **-Ms**. Pokud nezadáte hodnotu AVG_LO_SIZE, použije databázový server k výpočtu velikosti oblasti metadat výchozí hodnotu 8 kilobajtů.
- Velikost oblasti metadat určete ve volbě **-Ms** obslužného programu**onspaces**.

  Hodnotu, kterou je třeba zadat ve volbě **onspaces -Ms**, lze odhadnout pomocí postupu popsaného v části "Ruční úprava velikosti oblasti metadat pro nový blok" na stránce 6-12.

Další informace o monitorování využití místa v prostoru sbspace a o přidělování dalšího místa naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Ruční úprava velikosti oblasti metadat pro nový blok

V tomto postupu se předpokládá, že znáte velikost prostoru sbspace a potřebujete přidělit další prostor metadat. Každý blok může obsahovat metadata, ale celkový součet musí poskytovat dostatek prostoru pro všechny hlavičky LO (průměrná délka každé hlavičky je 570 bajtů) a seznam volných bloků (který obsahuje všechny volné oblasti v bloku), jak uvádí následující postup.

Ruční úprava velikosti oblasti metadat pro nový blok:

1. Volba **onstat -d** lze použít k získání velikosti aktuální oblasti metadat z pole **Velikost metadat**.
2. Odhadněte počet inteligentních velkých objektů, které by měly být uloženy v prostoru sbspace, a jejich průměrnou velikost.
3. Celkovou velikost oblasti metadat vypočítejte podle následujícího vzorce:

   ```
   Celková velikost metadat (kB) = ( početLO*570)/1024 +
                                   (nové_bloky*800) + 100
   ```

   početLO je předpokládaný počet inteligentních velkých objektů ve všech blocích prostorů sbspace včetně nového.

   nové_bloky je celkový počet bloků v prostoru sbspace.

   a. Chcete-li získat další požadovanou oblast metadat, odečtěte aktuální velikost oblasti metadat získanou v kroku 1 od hodnoty získané v kroku 3.

   b. Po přidání jiného bloku zadejte ve volbě **-Ms** příkazu **onspaces -a** hodnotu získanou v kroku 3a.

Příklad výpočtu oblasti metadat pro nový blok: Tento příklad umožňuje odhadnout velikost metadat vyžadovanou pro dva bloky prostoru sbspace pomocí předchozího postupu:

1. Předpokládejme, že pole **Velikost metadat** ve volbě **onstat -d** zobrazuje, že aktuální oblast metadat je 1000 stránek.

   Pokud je velikost stránky systému 2048 bajtů, je velikost této oblasti metadat 2000 kilobajtů, jak uvádí následující výpočet:

   ```
   aktuální metadata = (velikost_metadat * velikost_stránky) / 1024
                     = (1000 * 2048) / 1024
                    = 2000 kB
   ```
2. Předpokládejme, že chcete 31 000 inteligentních velkých objektů ve dvou blocích prostoru sbspace
3. Následující vzorec umožňuje vypočítat celkovou velikost oblasti metadat požadovanou pro oba bloky, zlomky se zaokrouhlují nahoru:

   ```
   Celková velikost metadat = (početLO*570)/1024 + (počet_bloků*800) + 100
                   = (31 000 * 570)/1024 + (2*800) + 100
                   = 17256 + 1600 + 100
                  = 18956 kB
   ```

4. Chcete-li získat další požadovanou oblast metadat, odečtěte aktuální velikost oblasti metadat získanou v kroku 1 od hodnoty získané v kroku 3.

```
Další metadata = Celková velikost metadat - Aktuální metadata
                       = 18956 - 2000
                      = 16956 kB
```

5. Pokud přidáte blok do prostoru sbspace, zadejte pomocí volby **-Ms** příkazu **onspaces -a** velikost oblasti metadat 16 956 kB.

```
% onspaces -a sbchk2 -p /dev/raw_dev1 -o 200 -Ms 16956
```

Další informace o volbě **onspaces** a **onstat -d** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Zlepšení vstupu - výstupu metadat inteligentních velkých objektů

Stránky metadat v prostoru sbspace obsahují informace o umístění inteligentních velkých objektů v prostoru sbspace. Tyto stránky jsou obvykle intenzivně čteny. Vstup - výstup těchto stránek můžete distribuovat následujícími způsoby:

- Zrcadlete bloky, které obsahují metadata.

  Další informace o důsledcích zrcadlení uvádí část “Zvážení možnosti použití zrcadlení pro komponenty kritických dat” na stránce 5-5,

- Umístěte stránky metadat na nejrychlejší část disku.

  Protože jsou stránky metadat obvykle nejintenzivněji čtenou částí prostoru sbspace, umístěte stránky metadat blíže ke středu disku, abyste snížili čas přístupu. Chcete-li umístit stránky metadat, použijte při vytváření prostoru sbspace nebo při přidání bloku pomocí obslužného programu **onspaces** volbu **-Mo**.

- Rozložte oblasti metadat mezi více disků.

  Chcete-li rozložit stránky metadat mezi více disků, vytvořte v prostoru sbspace více bloků, které jsou umístěny na oddělených discích. Pokud přidáte blok do prostoru sbspace pomocí obslužného programu **onspaces**, přidělte pomocí volby **-Ms** stránky pro informace metadat.

  I když se databázový server pokouší zachovat informace metadat s odpovídajícími daty ve stejném bloku, nelze zaručit, že zůstanou pohromadě.

- Zmenšete počet oblastí, které zabírají jednotlivé inteligentní velké objekty.

  Pokud inteligentní velký objekt zabírá více oblastí, obsahují metadata pro každou oblast oddělený deskriptor. Chcete-li snížit počet položek deskriptorů, které je třeba přečíst u každého inteligentního velkého objektu, zadejte při vytvoření inteligentního velkého objektu jeho očekávanou konečnou velikost.

  Pokud zadáte konečnou velikost jednou z následujících funkcí, přidělí databázový server inteligentnímu velkému objektu jedinou oblast (pokud má v bloku souvislé úložiště):

  – Funkce DataBlade API **mi_lo_specset_estbytes**

  – Funkce ESQL/C **ifx_lo_specset_estbytes**

  Další informace funkcích k otevření inteligentních velkých objektů a k nastavení odhadovaného počtu bajtů naleznete v příručkách *IBM Informix ESQL/C Programmer's Manual* a *IBM Informix DataBlade API Programmer's Guide*.

  Další informace o nastavení velikosti oblastí naleznete v části “Velikosti oblastí pro rozšíření prostoru sbspace” na stránce 5-19.

Důležité: Abyste dosáhli nejvyšší dostupnosti dat, je třeba zrcadlit všechny bloky prostorů sbspace, které obsahují metadata.

Monitorování prostorů sbspace

Lepšího výkonu vstupu - výstupu dosáhnete, pokud všechny inteligentní velké objekty přidělíte do jedné souvislé oblasti. Další informace o nastavení velikosti oblastí naleznete v části “Velikosti oblastí pro rozšíření prostoru sbspace” na stránce 5-19.

Souvislost poskytuje následující výhody z hlediska výkonu vstupu - výstupu:

- Minimalizuje pohyb diskové hlavy
- Požaduje ke čtení inteligentních velkých objektů méně operací vstupu - výstupu
- Při provádění rozsáhlých sekvenčních čtení lze využít výhody odlehčeného vstupu - výstupu, které čte větší bloky dat (60 kB nebo více, v závislosti na platformě) v jedné operaci vstupu - výstupu

Monitorování efektivity můžete provádět pomocí následujících obslužných programů příkazového řádku:

- **oncheck -cS, -pe** a **-pS**
- volba **onstat -g smb s**

Následující části popisují způsob použití těchto nástrojů k monitorování prostorů sbspace. Další informace o obslužném programu **oncheck** a **onstat** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Použití oblužného programu oncheck -cS

Volba **oncheck -cS** kontroluje oblasti inteligentního velkého objektu a oddíly prostoru sbspace v oblasti uživatelských dat. Obrázek 6-5 znázorňuje příklad výstupu volby**-cS** pro prostor **s9_sbspc**.

Hodnoty ve sloupcích **Sbs#**, **Chk#** a **Seq#** odpovídají hodnotě **Stránka bloku prostoru** ve výstupu **-pS**. Sloupce **Bajtů** a **Stránek** zobrazují velikost jednotlivých inteligentních velkých objektů v bajtech a stránkách.

Chcete-li vypočítat průměrnou velikost inteligentních velkých objektů, sečtěte hodnoty ve sloupcích **velikost (bajtů)** a vydělte tuto hodnotu počtem inteligentních velkých objektů. V Obrázek 6-5 je průměrný počet přidělených bajtů 2690, jak zobrazuje následující výpočet:

```
Průměrná velikost v bajtech = (15736 + 98 + 97 + 62 + 87 + 56) / 6
                            = 16136 / 6
                            = 2689,3
```

Další informace o tom, jak určit velikost inteligentních velkých objektů, aby se upravila velikost oblasti, naleznete v části “Velikosti oblastí pro rozšíření prostoru sbspace” na stránce 5-19.

```
Kontrola prostoru 's9_sbspc' ...

Velké objekty
  ID              Ref Velikost   Přiděleno      Příznak Poslední
  Sbs# Chk#  Seq#  Poč    (bajtů) stránek Oblasti vytvoření úprava
  ---- ---- -----  ---- ---------- -------- ----- ----- ------------------------
     2    2     1    1      15736        8     1 N-N-H Čtv Čer 21 16:59:12 2007
     2    2     2    1         98        1     1 N-K-H Čtv Čer 21 16:59:12 2007
     2    2     3    1         97        1     1 N-K-H Čtv Čer 21 16:59:12 2007
     2    2     4    1         62        1     1 N-K-H Čtv Čer 21 16:59:12 2007
     2    2     5    1         87        1     1 N-K-H Čtv Čer 21 16:59:12 2007
     2    2     6    1         56        1     1 N-K-H Čtv Čer 21 16:59:12 2007
```

Obrázek 6-5. Výstup příkazu oncheck -cS

Pole **Oblasti** zobrazuje minimální velikost oblastí v počtech stránek, přidaných jednotlivým inteligentním velkým objektům.

Použití příkazu oncheck -pe

Vykonáním obslužného programu **oncheck -pe** to lze zobrazit následující informace a určit, zda jsou inteligentní velké objekty umístěny v souvislém prostoru sbspace:

- Určuje jednotlivé inteligentní velké objekty ve výrazu SBLOBSpace LO

  Tři hodnoty v závorkách SBLOBSpace LO odpovídají sloupcům **Sbs#**, **Chk#** a **Seq#** ve výstupu **-cS**.
- Posun jednotlivých inteligentních velkých objektů
- Počet diskových stránek (*ne* prostorů sbpage) použitých jednotlivými inteligentními velkými objekty

Tip: Příkaz **oncheck -pe** poskytuje informace o využití prostorů sbspace, přičemž jednotkou jsou stránky databázového serveru, nikoli stránky sbpage.

Obrázek 6-6 znázorňuje ukázkový výstup. V tomto příkladu zobrazuje pole **velikost**, že první inteligentní velký objekt zabírá osm stránek. Protože pole **posun** zobrazuje, že první inteligentní velký objekt začíná na stránce 53 a druhý inteligentní objekt začíná na stránce 61.

```
Chunk Pathname                                      Size      Used      Free
                                                    1000       940        60

    Description                                        Offset     Size
    -------------------------------------------------- -------- --------
    RESERVED PAGES                                            0        2
    CHUNK FREELIST PAGE                                       2        1
    s9_sbspc:'informix'.TBLSpace                              3       50
    SBLOBSpace LO [2,2,1]                                    53        8
    SBLOBSpace LO [2,2,2]                                    61        1
    SBLOBSpace LO [2,2,3]                                    62        1
    SBLOBSpace LO [2,2,4]                                    63        1
    SBLOBSpace LO [2,2,5]                                    64        1
    SBLOBSpace LO [2,2,6]                                    65        1
...
```

Obrázek 6-6. Výstup příkazu oncheck -pe znázorňující využití souvislého prostoru

Použití oblužného programu oncheck -pS

Volba **oncheck -pS** zobrazí informace o oblastech inteligentních velkých objektů a oblastech metadat v oddílech prostoru sbspace. Pokud neurčíte název prostoru sbspace na příkazovém řádku, příkaz **oncheck** zkontroluje a zobrazí metadata všech prostorů sbspace. Obrázek 6-7 znázorňuje příklad výstupu volby **-pS** pro prostor **s9_sbspc**.

Chcete-li zobrazit informace o inteligentních velkých objektech, spusťte následující příkaz:

```
mncheck -pS název_prostoru
```

Výstup **oncheck -pS** zobrazuje pro jednotlivé inteligentní velké objekty v prostoru dbspace následující informace:

- Stránka bloku prostoru
- Velikost v bajtech jednotlivých inteligentních velkých objektů
- ID objektu, které používají funkce DataBlade API a ESQL/C
- Charakteristika úložiště jednotlivých inteligentních velkých objektů

Pokud k vytvoření prostor sbspace používáte příkaz **onspaces -c -S** , můžete použít k určení různých charakteristik inteligentních velkých objektů volbu **-Df** . Po vytvoření prostoru sbspace můžete použít ke změně atributů příkaz **onspaces -ch** . Pole **Příznaky vytvoření** ve

výstupu příkazu **oncheck -pS** zobrazuje tyto tři charakteristiky a další atributy jednotlivých inteligentních velkých objektů. Obrázek 6-7 znázorňuje pole **Příznaky vytvoření** hodnotu LO_LOG, protože značka LOGGING byla pomocí volby **-Df** nastavena na hodnotu ON.

```
Space Chunk Page = [2,2,2] Object ID = 987122917
  LO SW Version             4
  LO Object Version         1
  Created by Txid           7
  Flags               0x31 LO_LOG LO_NOKEEP_LASTACCESS_TIME LO_HIGH_INTEG
  Data Type                 0
  Extent Size               -1
  IO Size                   0
  Created                   Thu Apr 12 17:48:35 2001
  Last Time Modified        Thu Apr 12 17:48:43 2001
  Last Time Accessed        Thu Apr 12 17:48:43 2001
  Last Time Attributes Modified Thu Apr 12 17:48:43 2001
  Ref Count                 1
  Create Flags        0x31 LO_LOG LO_NOKEEP_LASTACCESS_TIME LO_HIGH_INTEG
  Status Flags              0x0 LO_FROM_SERVER
  Size (Bytes)              2048
  Size Limit                -1
  Total Estimated Size      -1
  Deleting TxId             -1
  LO Map Size               200
  LO Map Last Row           -1
  LO Map Extents            2
  LO Map User Pages         2
```

Obrázek 6-7. Výstup příkazu oncheck -pS

Použití příkazu onstat -g smb

Volba **onstat -g smb s** zobrazuje následující charakteristiky, které mají vliv na výkon vstupu - výstupu jednotlivých prostorů sbspace:

- Stav protokolování

  Pokud aplikace aktualizují dočasné inteligentní velké objekty, není protokolování vyžadováno. Pokud chcete snížit aktivitu vstupu - výstupu do logického protokolu, vytížení procesoru a paměťových zdrojů, můžete omezit množství aktivity logického protokolu.

- průměrnou velikost inteligentního velkého objektu,

  Za účelem omezení počtu operací vstupu - výstupu požadovaných ke čtení celého inteligentního velkého objektu by měly být průměrná velikost a velikost oblasti podobné. Pole výstupu **avg s/kb** zobrazuje průměrnou velikost inteligentního velkého objektu v kilobajtech. Obrázek 6-8 znázorňuje pole výstupu **avg s/kb** hodnotu 30 kilobajtů.

  Pomocí jedné z následujících funkcí zadejte konečnou velikost inteligentního velkého objektu, aby byl objekt přidělen jako jediná oblast:

  – Funkce DataBlade API **mi_lo_specset_estbytes**

  – Funkce ESQL/C **ifx_lo_specset_estbytes**

  Další informace funkcích k otevření inteligentních velkých objektů a k nastavení odhadovaného počtu bajtů naleznete v příručkách *IBM Informix ESQL/C Programmer's Manual* a *IBM Informix DataBlade API Programmer's Guide*.

- Velikost první oblasti, velikost další oblasti a minimální velikost oblasti

  Pole výstupu **1st sz/p**, **nxt sz/p** a **min sz/p** zobrazují tyto velikosti oblastí, pokud nastavíte ve volbě**-Df** příkazu **onspaces** značky oblasti. V Obrázek 6-8 tato pole výstupu zobrazují hodnoty 0 a -1, protože tyto značky nejsou nastaveny v příkazu **onspaces**.

```
sbnum 7    address 2afae48
  Space    : flags    nchk owner    sbname
             -------- 1    informix client
  Defaults : LO_LOG LO_KEEP_LASTACCESS_TIME

  LO     : ud b/pg flags   flags    avg s/kb max lcks
           2048    0       -------- 30       -1
  Ext/IO : 1st sz/p nxt sz/p min sz/p mx io sz
           4        0        0        -1

  HdrCache : max   free
             512   0
```

Obrázek 6-8. Výstup příkazu onstat -g smb s

Změna úložných charakteristik inteligentních velkých objektů

Pokud vytváříte prostor sbspace, ale nezadáte hodnoty ve volbě **-Df** příkazu **onspaces -c -S**, použijete pro charakteristiku a atributy úložiště (například protokolování a ukládání do vyrovnávací paměti) výchozí hodnoty.

Po monitorování prostorů sbspace můžete chtít změnit charakteristiku úložiště, stav protokolování, režim uzamčení a jiné atributy inteligentních velkých objektů.

Administrátor databáze nebo programátor může použít k přepsání těchto výchozích hodnot pro charakteristiku a atributy úložiště následující metody:

- Administrátor databáze může použít jednu z následujících voleb obslužného programu **onspaces**:
  - Při prvním vytvoření prostoru sbspace zadejte hodnoty pomocí příkazu **onspaces -c -S** .
  - Po vytvoření prostoru sbspace lze změnit hodnoty příkazem **onspaces -ch** .

  Tyto hodnoty zadejte ve volbách značek volby **-Df** příkazu **onspaces**. Další informace o nástroji **onspaces** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.
- Administrátor databáze může určit hodnoty v klauzuli PUT příkazů CREATE TABLE a ALTER TABLE.

  Tyto hodnoty přepisují hodnoty v nástroji **onspaces** a platí pouze pro inteligentní velké objekty uložené v přidruženém sloupci příslušné tabulky. Ve stejném prostoru sbspace mohou být také uloženy další inteligentní velké objekty (ze sloupců v jiných tabulkách) . Tyto jiné sloupce stále používají charakteristiky a atributy úložiště prostoru sbspace definované obslužným programem **onspaces** (nebo výchozí hodnoty, pokud nejsou definovány příkazem **onspaces**), pokud nejsou hodnoty přepsány pomocí klauzule PUT u jednotlivých sloupců.

  Pokud neurčíte pro sloupec inteligentního velkého objektu charakteristiku úložiště v klauzuli PUT, zdědí se charakteristika od prostoru sbspace.

  Pokud neurčíte klauzuli PUT při vytvoření sloupců inteligentních velkých objektů, bude databázový server ukládat inteligentní velké objekty do výchozího prostoru sbspace systému, který je určen konfiguračním parametrem SBSPACENAME. V tom případě jsou charakteristiky a atributy úložiště zděděny z prostoru sbspace SBSPACENAME.

DB-Access

- K úpravě charakteristiky úložiště sloupce inteligentního velkého objektu mohou programátoři používat funkce v DataBlade API a ESQL/C.

Jazyk ESQL/C

Informace o funkcích rozhraní DataBlade API pro práci s inteligentními velkými objekty naleznete v příručce *IBM Informix DataBlade API Programmer's Guide*. Informace o funkcích rozhraní ESQL/C pro práci s inteligentními velkými objekty naleznete v příručce *IBM Informix ESQL/C Programmer's Manual*.

Konec Jazyk ESQL/C

Konec DB-Access

Tabulka 6-1 popisuje způsoby, jak upravovat charakteristiky úložiště inteligentních velkých objektů.

Tabulka 6-1. Úprava charakteristiky a jiných atributů úložiště

Úložiště: Charakteristika nebo atribut	**Výchozí nastavení systému Hodnota**	**Charakteristika úložiště určená systémem pomocí volby -Df obslužného programu onspaces**	**Úroveň sloupce Úložiště Charakteristika určená klauzulí PUT příkazu CREATE TABLE nebo ALTER TABLE**	**Charakteristika úložiště určená pomocí funkce rozhraní DataBlade API**	**Charakteristika úložiště určená pomocí funkce ESQL/C**
Čas posledního přístupu	VYPNUTO	ČAS PŘÍSTUPU	ZACHOVAT ČAS PŘÍSTUPU, NEZACHOVAT ČAS PŘÍSTUPU	Ano	Ano
Režim uzamčení	BLOB	REŽIM_UZAMČENÍ	Ne	Ano	Ano
Stav protokolování	VYPNUTO	PROTOKOLOVÁNÍ	PROTOKOL, BEZ PROTOKOLU	Ano	Ano
Velikost oblasti	Žádný	VELIKOST_ OBLASTI	VELIKOST_ OBLASTI	Ano	Ano
Velikost další oblasti	Žádný	DALŠÍ_VELIKOST	Ne	Ne	Ne
Minimální velikost oblasti	2 kB v systému Windows 4 kB v systému UNIX	MINIMÁLNÍ_ VELIKOST_ OBLASTI	Ne	Ne	Ne
Velikost inteligentního velkého objektu	8 kB	Průměrná velikost všech inteligentních velkých objektů v prostoru sbspace: AVG_LO_SIZE	Ne	Odhadovaná velikost určitého inteligentního velkého objektu Maximální velikost určitého inteligentního velkého objektu	Odhadovaná velikost určitého inteligentního velkého objektu Maximální velikost určitého inteligentního velkého objektu
Využití společné oblasti vyrovnávacích pamětí	ZAPNUTO	UKLÁDÁNÍ DO VYROVNÁVACÍ PAMĚTI	Ne	příznaky LO_BUFFER a LO_ NOBUFFER	příznaky LO_BUFFER a LO_ NOBUFFER

Tabulka 6-1. Úprava charakteristiky a jiných atributů úložiště (pokračování)

Úložiště: Charakteristika nebo atribut	**Výchozí nastavení systému Hodnota**	**Charakteristika úložiště určená systémem pomocí volby -Df obslužného programu onspaces**	**Úroveň sloupce Úložiště Charakteristika určená klauzulí PUT příkazu CREATE TABLE nebo ALTER TABLE**	**Charakteristika úložiště určená pomocí funkce rozhraní DataBlade API**	**Charakteristika úložiště určená pomocí funkce ESQL/C**
Název prostoru sbspace	NÁZEV-PROSTORU-SBSPACE	Není ve volbě **-Df**. Název určený ve volbě **onspaces -S** .	Název existujícího prostoru sbspace, ve kterém je uložen inteligentní velký objekt: klauzule PUT ... IN	Ano	Ano
Fragmentace mezi více prostory sbspace	Žádný	Ne	Schéma distribuce cyklické obsluhy: klauzule PUT ... IN	Schéma distribuce cyklické obsluhy nebo schéma distribuce založené na výrazu	Schéma distribuce cyklické obsluhy nebo schéma distribuce založené na výrazu
Čas posledního přístupu	VYPNUTO	ČAS PŘÍSTUPU	ZACHOVAT ČAS PŘÍSTUPU, NEZACHOVAT ČAS PŘÍSTUPU	Ano	Ano
Režim uzamčení	BLOB	REŽIM_UZAMČENÍ	Ne	Ano	Ano
Stav protokolování	VYPNUTO	PROTOKOLOVÁNÍ	PROTOKOL, BEZ PROTOKOLU	Ano	Ano
Velikost oblasti	Žádný	VELIKOST_ OBLASTI	VELIKOST_ OBLASTI	Ano	Ano
Velikost další oblasti	Žádný	DALŠÍ_VELIKOST	Ne	Ne	Ne
Minimální velikost oblasti	2 kB v systému Windows 4 kB v systému UNIX	MINIMÁLNÍ_ VELIKOST_ OBLASTI	Ne	Ne	Ne
Velikost inteligentního velkého objektu	8 kB	Průměrná velikost všech inteligentních velkých objektů v prostoru sbspace: AVG_LO_SIZE	Ne	Odhadovaná velikost určitého inteligentního velkého objektu Maximální velikost určitého inteligentního velkého objektu	Odhadovaná velikost určitého inteligentního velkého objektu Maximální velikost určitého inteligentního velkého objektu
Využití společné oblasti vyrovnávacích pamětí	ZAPNUTO	UKLÁDÁNÍ DO VYROVNÁVACÍ PAMĚTI	Ne	příznaky LO_BUFFER a LO_ NOBUFFER	příznaky LO_BUFFER a LO_ NOBUFFER
Název prostoru sbspace	NÁZEV-PROSTORU-SBSPACE	Není ve volbě **-Df**. Název určený ve volbě **onspaces -S** .	Název existujícího prostoru sbspace, ve kterém je uložen inteligentní velký objekt: klauzule PUT ... IN	Ano	Ano

Tabulka 6-1. Úprava charakteristiky a jiných atributů úložiště (pokračování)

Úložiště: Charakteristika nebo atribut	Výchozí nastavení systému Hodnota	Charakteristika úložiště určená systémem pomocí volby -Df obslužného programu onspaces	Úroveň sloupce Úložiště Charakteristika určená klauzulí PUT příkazu CREATE TABLE nebo ALTER TABLE	Charakteristika úložiště určená pomocí funkce rozhraní DataBlade API	Charakteristika úložiště určená pomocí funkce ESQL/C
Fragmentace mezi více prostory sbspace	Žádný	Ne	Schéma distribuce cyklické obsluhy: klauzule PUT ... IN	Schéma distribuce cyklické obsluhy nebo schéma distribuce založené na výrazu	Schéma distribuce cyklické obsluhy nebo schéma distribuce založené na výrazu

Úprava sloupců inteligentních velkých objektů

Při vytváření tabulky máte pro konkrétní sloupce inteligentních velkých objektů k dispozici následující možnosti výběru charakteristiky a dalších atributů úložiště (například stav protokolování, ukládání do vyrovnávací paměti a režim uzamčení):

- Použijte hodnoty nastavené při vytvoření prostoru sbspace. Tyto hodnoty jsou určeny jedním z následujících způsobů:
  - Pomocí různých značek volby **-Df** příkazu **onspaces -c -S**
  - Pomocí výchozí hodnoty systému pro libovolnou konkrétní značku, která nebyla určena

  Návod ke změně výchozí charakteristiky úložiště značek **-Df** naleznete v části “Volby obslužného programu onspaces, které ovlivňují vstup - výstup prostoru sbspace” na stránce 5-19.
- Klauzuli PUT příkazu CREATE TABLE použijte k zadání jiných než výchozích hodnot určitých charakteristik nebo atributů.

  Pokud v klauzuli PUT neurčíte konkrétní charakteristiky a atributy, použijí se v příkazu výchozí hodnoty. onspaces -c -S

Později můžete ke změně volitelných charakteristik úložiště těchto sloupců použít klauzuli PUT příkazu ALTER TABLE. Tabulka 6-1 na stránce 6-18 znázorňuje charakteristiky a atributy, které můžete změnit.

Klauzuli PUT příkazu ALTER TABLE lze použít k následujícím akcím:

- Při přidání sloupce do tabulky zadejte charakteristiku a paměťové místo inteligentního velkého objektu.

  Inteligentní velké objekty v nových sloupcích mohou mít jiné charakteristiky než objekty v existujících sloupcích.
- Změňte charakteristiku inteligentního velkého objektu existujícího sloupce.

  Nová charakteristika sloupce platí pouze pro nové inteligentní velké objekty vytvořené v tomto sloupci. Charakteristika existujících inteligentních velkých objektů zůstává stejná.
- Jednoduché velké objekty lze převést na inteligentní velké objekty změnou typu sloupce z typu TEXT na CLOB nebo z typu BYTE na BLOB. Další informace o převodu jednoduchých velkých objektů naleznete v příručce *IBM Informix Migration Guide*.

Data BLOB v tabulce **catalog** v databázi **superstores_demo** jsou například uložena v prostoru **s9_sbspc** s vypnutým protokolováním a velikostí oblasti 100 kB. Chcete-li zapnout protokolování nebo uložit inteligentní velké objekty do jiného prostoru sbspace, můžete použít klauzuli PUT příkazu ALTER TABLE. Další informace o protokolování prostoru sbspace naleznete v části o inteligentních velkých objektech v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Další informace o změně oblastí prostoru sbspace pomocí příkazu CREATE TABLE naleznete v části "Velikosti oblastí inteligentních velkých objektů v prostorech sbspace" na stránce 6-22.

Další informace o příkazech CREATE TABLE a ALTER TABLE naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Správa oblastí

Při přidávání řádků do tabulky databázový server přiděluje místo na disku v jednotkách nazývaných *oblasti*. Každá oblast je blok fyzicky souvislých stránek z prostoru dbspace. I když prostor dbspace obsahuje více než jeden blok, je každá oblast přidělena pouze v jednom bloku, aby zůstala souvislá.

Souvislost má významný vliv na výkon. Pokud jsou datové stránky souvislé a pokud při dopředném čtení, při použití odlehčeného prohledávání nebo odlehčených operací vstupu - výstupu čte databázový server řádky sekvenčně, je pohyb hlavy disku minimální. Další informace o těchto operacích naleznete v částech "Sekvenční prohledávání" na stránce 5-24, "Odlehčené prohledávání" na stránce 5-24 a "Konfigurační parametry, které ovlivňují vstup - výstup prostoru sbspace" na stránce 5-18.

Mechanismus oblastí je kompromisem mezi následujícími protichůdnými požadavky:

- Většina prostorů dbspace je sdílena mezi několika tabulkami.
- Velikost těchto tabulek není předem známa.
- Tabulky se mohou zvětšovat v libovolnou dobu libovolnou rychlostí.
- Za účelem lepšího výkonu by měly všechny stránky tabulky sousedit.

Protože velikost tabulek není známa, nemůže databázový server předem přidělovat místo pro tabulku. Proto přidává databázový server oblasti pouze tehdy, pokud jsou třeba, ale všechny stránky v jedné oblasti jsou souvislé, aby byl zajištěn vyšší výkon. Dále pokud databázový server vytvoří novou oblast, která sousedí s předchozí oblastí, bude server považovat obě oblasti za jedinou oblast.

Volba velikostí oblastí tabulek

Při vytváření tabulky musíte zadat velikosti oblasti pro následující paměťové prostory:

- Datové řádky tabulky v prostoru dbspace
- Každý fragment ve fragmentované tabulce
- Inteligentní velké objekty v prostoru sbspace

Velikosti oblastí pro tabulky v prostoru dbspace

Při vytvoření tabulky můžete zadat velikost první oblasti a velikost oblastí přidávaných při zvětšování tabulky. Následující příklad vytváří tabulku s počáteční oblastí 512 kB a přidávanými oblastmi velikosti 100 kB:

```
CREATE TABLE big_one (...specifikace sloupců...)
   IN big_space
   EXTENT SIZE 512
   NEXT SIZE 100
```

Výchozí velikost oblasti a velikost další oblasti je osmkrát větší, než velikost diskové stránky v systému. Pokud má stránka například velikost 2 kB, výchozí délka je 16 kB.

Ke změně velikosti přidávaných oblastí použijte příkaz ALTER TABLE. Tato změna nemá vliv na již existující oblasti. Následující příklad mění velikost další oblasti tabulky na 50 kB:

```
ALTER TABLE big_one MODIFY NEXT SIZE 50
```

Na výkon nemá významný vliv velikost dalších oblastí následujících typů tabulek:

- Malá tabulka je definována jako tabulka, která má pouze jednu oblast. Pokud je taková tabulka často používána, zůstávají velké části tabulky v paměti.
- Tabulka, která se používá občas, není z hlediska výkonu důležitá bez ohledu na její velikost.
- Tabulka, která se nachází ve vyhrazeném prostoru dbspace, vždy obdrží nové oblasti, které sousedí se starými oblastmi. Velikost těchto oblastí není důležitá, protože jsou sousední a chovají se jako jedna oblast.

Pokud těmto typům tabulek přiřazujete velikost oblasti, je jediným důvodem, aby se zabránilo vytváření velkého počtu oblastí. Velký počet oblastí způsobuje, že hledání dat trvá databázovému serveru déle. Dále existuje horní limit počtu možných oblastí. Část ("Úvahy o horním limitu oblastí" na stránce 6-24 se zabývá tímto tématem.)

Na velikost oblasti neexistuje žádný limit kromě velikosti bloku. Maximální velikost bloku je 4 TB. Pokud znáte konečnou velikost tabulky (dokážete její velikost spolehlivě odhadnout na 25 procent), alokujte celou její velikost v počáteční oblasti. Pokud tabulky stabilně rostou do neznámé velikosti, přiřaďte jim velikosti další oblasti tak, aby mohly sdílet prostor dbspace s menším počtem oblastí. Jeden možný přístup popisují následující kroky.

Přidělení prostoru pro oblasti tabulky:

1. Rozhodněte se, jak rozdělíte prostor mezi tabulky.

   Prostor dbspace můžete rozdělit mezi tři tabulky například v poměru 0,4:0,2:0.3 (10 procent bude vyhrazeno na malé tabulky a režii).
2. Jako počáteční oblast přidělte každé tabulce jednu čtvrtinu jejího podílu prostoru dbspace.
3. Jako velikost další oblasti přidělte každé tabulce jednu osminu jejího podílu.
4. Zvětšování tabulky můžete monitorovat běžným způsobem pomocí příkazu **oncheck**.

Během zaplňování prostoru dbspace nemusíte mít k dispozici dostatek volného souvislého místa pro vytvoření oblasti určené velikosti. V takovém případě přidělí databázový server největší možnou dostupnou souvislou oblast.

Velikosti oblastí pro fragmenty tabulek

Pokud fragmentujete existující tabulku, můžete chtít upravit velikost další oblasti, protože každý fragment vyžaduje méně místa, než původní nefragmentovaná tabulka. Pokud byla u nefragmentované tabulky definována velká velikost další oblasti, používá databázový server stejnou velikost pro další oblasti *všech* fragmentů, což vede k nadbytečnému přidělování místa na disku. Jednotlivé fragmenty vyžadují pouze část prostoru pro celou tabulku.

Pokud například fragmentujete předchozí ukázkovou tabulku **big_one** na pět disků, můžete velikost další oblasti upravit na jednu pětinu původní velikosti. Další informace o příkazu ALTER FRAGMENT naleznete v příručce *IBM Informix Guide to SQL: Syntax*. Následující příklad mění velikost další oblasti na jednu pětinu původní velikosti:

```
ALTER TABLE big_one MODIFY NEXT SIZE 20
```

Velikosti oblastí inteligentních velkých objektů v prostorech sbspace

Při vytváření tabulky je doporučeno použít pro inteligentní velké objekty v prostoru sbspace jednu z následujících velikostí oblasti:

- Velikost oblasti vypočtená databázovým serverem

- Konečná velikost inteligentního velkého objektu, jak je při otevření prostoru sbspace v aplikaci označena jednou z následujících funkcí:

DB-Access

– Funkce DataBlade API **mi_lo_specset_estbytes**

Další informace o funkcích rozhraní DataBlade API pro otevření inteligentního velkého objektu a nastavení odhadovaného počtu bajtů naleznete v příručce *IBM Informix DataBlade API Programmer's Guide*.

Konec DB-Access

Jazyk ESQL/C

– Funkce ESQL/C **ifx_lo_specset_estbytes**

Další informace o funkcích jazyka ESQL/C pro otevření inteligentního velkého objektu a nastavení odhadovaného počtu bajtů naleznete v příručce *IBM Informix ESQL/C Programmer's Manual*.

Konec Jazyk ESQL/C

Další informace o nastavení velikosti oblastí naleznete v části “Velikosti oblastí pro rozšíření prostoru sbspace” na stránce 5-19. Další informace naleznete v části“Monitorování prostorů sbspace” na stránce 6-14.

Monitorování aktivních prostorů tblspace

Monitorováním prostorů tblspace můžete zjistit, které tabulky jsou aktivní. Aktivní jsou ty tabulky, jejichž jednotkový proces byl aktuálně otevřen.

Výstup z volby **onstat -t** zahrnuje číslo prostoru tblspace a následující čtyři sloupce.

Pole	Popis
npages	Počet stránek přidělených prostoru tblspace
nused	Počet stránek použitých z této přidělené společné oblasti
nextns	Počet použitých oblastí
npdata	Počet použitých datových stránek

Pokud určitá operace potřebuje více stránek, než je k dispozici (**npages** minus **nused**), je vyžadována nová oblast. Pokud je v tomto bloku k dispozici dostatek místa, přidělí databázový server oblast v tomto bloku, pokud není, hledá databázový server místo v jiných dostupných blocích. Pokud žádný blok neobsahuje postačující souvislý prostor, použije databázový server největší blok nebo souvislý prostor, který lze v prostoru dbspace nalézt. Obrázek 6-9 znázorňuje příklad výstupu této volby.

```
Tblspaces
 n address  flgs ucnt tblnum   physaddr npages nused  npdata nrows  nextns
 0 422528   1    1    100001   10000e   150    124    0      0      3
 1 422640   1    1    200001   200004   50     36     0      0      1
54 426038   1    6    100035   1008ac   3650   3631   3158   60000  3
62 4268f8   1    6    100034   1008ab   8      6      4      60     1
63 426a10   3    6    100036   1008ad   368    365    19     612    3
64 426b28   1    6    100033   1008aa   8      3      1      6      1
193 42f840  1    6    10001b   100028   8      5      2      30     1
 7 active, 200 total, 64 hash buckets
```

Obrázek 6-9. Výstup příkazu onstat -t

Monitorování horního limitu oblastí a prokládání oblastí

Tato část popisuje následující témata:

- Úvahy o horním limitu počtu oblastí
- Kontrola prokládání oblastí
- Odstranění prokládání oblastí

Úvahy o horním limitu oblastí

Nedovolte, aby tabulka získala velký počet oblastí, protože existuje horní limit počtu možných oblastí. Pokus o přidání oblasti po dosažení limitu způsobí chybu -136 (Žádné další oblasti) po požadavku INSERT.

Zjištění horního limitu možného počtu oblastí pro tabulku:

1. Spuštěním volby **oncheck** získejte fyzickou adresu objektu (tabulka nebo fragment indexu), pro který chcete vypočítat omezení oblastí.

   ```
   oncheck -pt název_databáze:tabulkanázev
   ```

   Obrázek 6-10 znázorňuje ukázku výstupu příkazu **oncheck -pt**.

```
TBLspace Report for stores7:wbyrne.sfe_enquiry
Physical Address 7:711
Number of special columns 18
Number of keys 0
Number of extents 65
Number of data pages 960
```

Obrázek 6-10. Výstup příkazu oncheck -pt

1. Fyzickou adresu fragmentu tabulky nebo indexu rozdělte na číslo bloku (číslice před dvojtečkou) a číslo stránky (posledních pět číslic vpravo od dvojtečky) a pak spusťte příkaz **oncheck -pP** *id_bloku:id_stránky*.

   Obrázek 6-10 udává hodnotu **Physical Address** rovnou 7:711. V následujícím příkazu **oncheck** je proto hodnota **chunk_id** rovna 7 a hodnota **page_id** rovna 711:

   ```
   oncheck -pP 7:711
   ```

   Obrázek 6-11 znázorňuje ukázkový výstup tohoto příkazu.

```
addr   stamp  nslots flag type  frptr frcnt next prev
7:711 112686 5      2    PARTN 828   1196  0    0
slot ptr len flg
1     24  92   0
2    116  40   0
3    156 144   0
4    300   0   0
5    300 528   0
```

Obrázek 6-11. Výstup příkazu oncheck -pp číslo_bloku číslo_stránky

2. Ve výstupu příkazu **oncheck -pP** vezměte číslo ve sloupci **frcnt** a vydělte je číslem 8, získáte počet dalších oblastí, které jsou pro tento objekt k dispozici.

   V ukázkovém příkazu **oncheck -pP**, který znázorňuje Obrázek 6-11, je ve sloupci **frcnt** uvedena hodnota 1196. Následující výpočet udává počet dalších oblastí:

   ```
   Additional_extents = trunc (frcnt / 8)
                      = trunc (1196 / 8)
                      = 149
   ```

3. Chcete-li získat maximální počet oblastí, přičtěte k hodnotě Additional_extents hodnotu z řádku **Number of Extents** ve výstupu příkazu **oncheck -pt** podle následujícího vzorce:

   ```
   Maximum_number_extents = Additional_extents +
      Number_of_extents
   ```

   V ukázkovém příkazu **oncheck -pt**, který znázorňuje Obrázek 6-10, je v řádku **Number of extents** uvedena hodnota 65. Maximální počet oblastí této tabulky zobrazuje následující výpočet:

   ```
   Maximum_number_extents = 149 + 65 = 214
   ```

Aby se zajistilo, že nebude limit překročen, provádí databázový server následující akce:

- Databázový server kontroluje počet oblastí vždy, když vytváří novou oblast. Pokud je číslo vytvářené oblasti násobkem čísla 16, databázový server automaticky dvakrát zvětší velikost další oblasti této tabulky. Z toho důvodu databázový server při každém šestnáctém vytvoření dvakrát zvětší velikost další oblasti.
- Pokud databázový server vytváří oblast, která sousedí s předchozí oblastí, považuje obě oblasti za jedinou oblast.

Kontrola prokládání oblastí

Pokud dvě nebo více rostoucích tabulek sdílí prostor dbspace, lze oblasti z jednoho prostoru tblspace umístit mezi oblasti jiného prostoru tblspace. Pokud dojde k této situaci, budou se oblasti nazývat *prokládané*. Prokládání mezi oblastmi tabulky vytváří mezery, jak znázorňuje Obrázek 6-12. Pokud je třeba při hledání tabulky na disku číst více než jednu oblast, zejména při sekvenčním prohledávání, zhoršuje se výkon.

Obrázek 6-12. Prokládané oblasti tabulek

Pokuste se optimalizovat velikosti oblastí tabulky tak, aby se přiděloval souvislý prostor na disku, což omezuje pohyby hlav. Také zvažte umístění tabulek do oddělených prostorů dbspace.

Prokládání oblastí pravidelně kontrolujte pomocí monitorování bloků. Fyzické rozvržení informací v bloku můžete získat provedením příkazu **oncheck -pe**. Zobrazí se následující informace:

- Název a vlastník prostoru dbspace
- Počet bloků v prostoru dbspace
- Sekvenční rozvržení tabulek a volné místo v jednotlivých blocích
- Počet stránek vyhrazených jednotlivým oblastem tabulky nebo volné místo

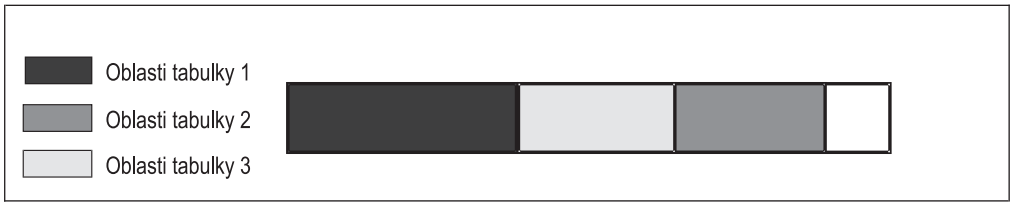
Tento výstup je vhodný ke zjištění stupně prokládání oblastí. Pokud databázový server nemůže v bloku přidělit oblast, přestože je k dispozici dostatečný počet volných stránek, je možné, že v bloku existuje špatné prokládání.

Odstranění prokládaných oblastí

Prokládané oblasti můžete odstranit jednou z následujících metod:

- Reorganizace tabulky pomocí příkazů UNLOAD a LOAD
- Vytvoření nebo úprava indexu v klastru
- Použití příkazu ALTER TABLE

Reorganizace prostorů dbspace a tabulek, aby se zabránilo prokládání oblastí:
Prokládané oblasti můžete odstranit opětovným vytvořením prostoru dbspace, aby byly oblasti jednotlivých tabulek opět souvislé, jak znázorňuje Obrázek 6-13. Pořadí reorganizovaných tabulek v prostoru dbspace není důležité, ale stránky jednotlivých reorganizovaných tabulek musí být souvislé, aby při sekvenčním čtení tabulky nebylo třeba nebylo třeba dlouhého vyhledávání. Pokud disková hlava čte tabulku nesekvenčně, pohybuje se pouze v prostoru, který zabírá tabulka.



Obrázek 6-13. Reorganizace prostoru dbspace za účelem odstranění prokládaných oblastí

Reorganizace tabulek v prostoru dbspace:

DB-Access

1. Tabulky v prostoru dbspace lze zkopírovat individuálně na pásku pomocí příkazu UNLOAD v nástroji DB–Access.

Konec DB-Access

2. Vypusťte všechny tabulky z prostoru dbspace.
3. Tabulky vytvořte znovu pomocí příkazu LOAD nebo obslužného programu **dbload**.

Příkaz LOAD znovu vytvoří tabulky se stejnými vlastnostmi jako dříve včetně stejných velikostí oblastí.

Tabulku také můžete uvolnit pomocí obslužného programu **onunload** a ve spojení s nástrojem **onload** tabulku znovu načíst. Další informace o výběru vhodného obslužného programu nebo příkazu naleznete v příručce *IBM Informix Migration Guide*.

Vytvoření nebo úprava indexu v klastru: V závislosti na okolnostech můžete odstranit prokládání oblastí, pokud vytvoříte klastrovaný index nebo změníte index na klastr. Pokud použijete klauzuli TO CLUSTER příkazu CREATE INDEX nebo ALTER INDEX, seřadí databázový server tabulku a znovu ji sestaví. Klauzule TO CLUSTER mění pořadí řádků ve fyzické tabulce, aby odpovídaly pořadí v indexu. Další informace naleznete v části "Klastrování" na stránce 7-9.

Klauzule TO CLUSTER odstraňuje prokládané oblasti:

- Blok musí obsahovat souvislý prostor, který je dostatečně velký pro opětovné vytvoření jednotlivých tabulek.
- Databázový server musí k opětovnému vytvoření tabulky použít souvislý prostor.

  Pokud před tímto souvislým prostorem existují bloky volného prostoru, může databázový server nejdříve přidělovat menší bloky. Databázový server přiděluje prostor pro proces ALTER INDEX od začátku bloku a hledá bloky volného prostoru s velikostí větší nebo rovnou velikosti určené pro další oblast. Pokud databázový server znovu vytvoří tabulku s menšími bloky volného prostoru, které jsou rozptýleny v bloku, neodstraní prokládání oblastí.

Chcete-li zobrazit umístění a velikost bloků volného prostoru, proveďte příkaz **oncheck -pe** .

Použití klauzule TO CLUSTER příkazu ALTER INDEX:

1. U všech tabulek v bloku vypusťte všechny fragmentované a odpojené indexy kromě toho, který chcete umístit na klastr.
2. Pomocí klauzule TO CLUSTER příkazu ALTER INDEX umístěte na klastr zbývající index.

   Pokud tabulku znovu vytvoříte pomocí opětovného uspořádání řádků, odstraní tento krok prokládání oblastí.
3. Vytvořte znovu všechny ostatní indexy.

   Indexy můžete v tomto kroku zhustit, protože databázový server řadí hodnoty indexu před jejich přidáním do B-stromu.

Před umístěním na klastr index nemusíte vypustit. Proces ALTER INDEX je však rychlejší než CREATE INDEX, protože databázový server čte datové řádky v pořadí klastru pomocí indexu. Výsledné indexy jsou navíc hustější.

Chcete-li zabránit v opakování problému, zvažte zvětšení velikosti oblastí tblspace. Další informace naleznete v příručce *IBM Informix Guide to SQL: Tutorial*.

Použití příkazu ALTER TABLE k odstranění prokládání oblastí: Pokud používáte příkaz ALTER TABLE k přidání nebo k vypuštění sloupce nebo ke změně datového typu sloupce, zkopíruje databázový server tabulku a znovu ji vytvoří. Pokud databázový server znovu vytvoří celou tabulku, přepíše tabulku do jiných oblastí prostoru dbspace. Pokud však jsou v prostoru dbspace jiné tabulky, nelze zaručit, že spolu budou nové oblasti sousedit.

Důležité: Databázový server během operace ALTER TABLE pro určité typy operací, které určíte v klauzulích ADD, DROP a MODIFY, nekopíruje ani nevytváří tabulku znovu. V těchto případech databázový server používá algoritmus změn na místě k aktualizaci jednotlivých řádků (spíše než během operace ALTER TABLE). Další informace o podmínkách tohoto algoritmu změn na místě naleznete v části"Změny na místě" na stránce 6-33.

Uvolnění nepoužitého prostoru oblasti

Jakmile databázový server přidělí prostor na disku prostoru tblspace jako část oblasti, zůstane tento prostor vyhrazen prostoru tblspace. I když budou všechny stránky oblasti po odstranění dat prázdné, nebude prostor na disku k dispozici pro použití jinými tabulkami.

Důležité: Pokud odstraníte řádky tabulky, použije databázový server tento prostor ke vložení nových řádků do té samé tabulky. Tato část popisuje postupy uvolnění nepoužitého prostoru pro použití jinými tabulkami.

Velikost tabulky můžete chtít změnit v případě, že tabulka nepotřebuje celý prostor, který jí byl původně přidělen. Můžete znovu přidělit menší prostor dbspace a nepotřebný prostor uvolnit k použití jinými tabulkami.

Jako administrátor databázového serveru můžete uvolnit diskový prostor v prázdných oblastech a zpřístupnit jej ostatním uživatelům. K opětovnému vytvoření tabulky můžete použít libovolný z následujících příkazů SQL:

- ALTER INDEX
- UNLOAD a LOAD
- ALTER FRAGMENT

Uvolnění prostoru v prázdné oblasti pomocí příkazu ALTER INDEX

Pokud tabulka s prázdnými oblastmi obsahuje index, můžete provést příkaz ALTER INDEX s klauzulí TO CLUSTER. Klastrování indexu znovu vytvoří tabulku v jiném umístění v prostoru dbspace. Všechny oblasti přidružené k předchozí verzi tabulky se uvolní. Nově vytvořená verze tabulky také nemá prázdné oblasti.

Další informace o syntaxi příkazu ALTER INDEX naleznete v příručce *IBM Informix Guide to SQL: Syntax*. Další informace o klastrování naleznete v části “Klastrování” na stránce 7-9.

Uvolnění prostoru v prázdné oblasti pomocí příkazů UNLOAD a LOAD

Pokud tabulka neobsahuje index, můžete tabulku uvolnit, znovu ji vytvořit (buď ve stejném prostoru dbspace, nebo v jiném) a znovu zavést data pomocí nástrojů **onunload** a **onload**.

Další informace o výběru správného obslužného programu nebo příkazu naleznete v příručce *IBM Informix Migration Guide*. Další informace o syntaxi příkazů UNLOAD a LOAD naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Uvolnění prostoru v prázdné oblasti pomocí příkazu ALTER FRAGMENT

K opětovnému vytvoření tabulky můžete použít příkaz ALTER FRAGMENT, který uvolňuje prostor oblastí přidělených této tabulce. Další informace o syntaxi příkazu ALTER FRAGMENT naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Správa uvolnění oblastí pomocí klíčového slova TRUNCATE

Příkaz TRUNCATE je klíčové slovo jazyka SQL, které rychle odstraní aktivní řádky z tabulek a struktur B-stromů jejich indexů bez vypuštění tabulky a jejího schématu, přístupových oprávnění, spouštěčů, omezení a jiných atributů. Pomocí tohoto příkazu jazyka SQL pro definici dat můžete vyprázdnit místní tabulku a znovu ji použít bez jejího opětovného vytvoření nebo můžete uvolnit úložný prostor, který uchovával její řádky a struktury B-stromu.

Existují dvě implementace příkazu TRUNCATE:

- První implementace nazývaná "rychlé zkrácení" pracuje s většinou tabulek.
- Druhá implementace nazývaná "pomalé zkrácení" pracuje s tabulkami, které obsahují průhledné datové typy, datové typy inteligentních velkých objektů nebo zděděné indexy definované na typech ROW v hierarchiích datových typů.

Výkonnostní výhody použití příkazu TRUNCATE TABLE místo příkazu DELETE jsou o mnoho lepší v implementaci rychlého zkrácení, protože tato implementace neprohlíží ani nespouští žádné řádky tabulky. K implementaci pomalého zkrácení dochází u tabulek, které obsahují průhledné datové typy, datové typy inteligentních velkých objektů nebo zděděné indexy definované na typech ROW v hierarchiích datových typů, protože operace zkrácení prohlíží každý řádek obsahující tyto položky.

Další informace o příkazu TRUNCATE naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Ukládání více fragmentů do jednoho prostoru dbspace

Do jednoho prostoru dbspace můžete uložit více fragmentů tabulky nebo indexu, a tím omezit celkový počet prostorů dbspace, potřebných pro fragmentovanou tabulku. Každý fragment je v prostoru dbspace uložen v samostatném pojmenovaném oddílu. Uložení fragmentů více tabulek nebo indexů do jednoho prostoru dbspace zjednodušuje správu prostorů dbspace.

Tuto funkci lze také použít ke zlepšení výkonu dotazů oproti uložení každého fragmentu do jiného prostoru dbspace, pokud je prostor dbspace uložen v rychlejším zařízení.

Další informace naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server* v části týkající se správy oddílů.

Změna tabulek

Existující tabulku můžete chtít změnit z různých důvodů:

- Chcete-li periodicky obnovovat velké tabulky s podporou rozhodování
- K přidání nebo vypuštění starých dat z určitého časového období
- K přidání, vypuštění nebo úpravě sloupců ve velkých tabulkách s podporou rozhodování, pokud vzniká potřeba různé analýzy dat

Volbu **onstat -g opn** můžete použít k zobrazení seznamu tabulek a oddílů indexu podle ID jednotkových procesů aktuálně otevřených v systému. Další informace naleznete v příručce *IBM Informix Administrator's Reference*.

Zavedení a uvolnění tabulek

Databáze pro aplikace s podporou rozhodování jsou obvykle vytvořeny periodicky zaváděnými tabulkami, které byly uvolněny z aktivních databází OLTP. Velké tabulky můžete rychle zavést pomocí jedné nebo více následujících metod:

- High-Performance Loader (zavaděč HPL)

  K rychlému zavedení tabulek můžete použít Zavaděč HPL v expresním režimu. Další informace o tom, jak databázový server provádí vysoce výkonné zavádění, naleznete v příručce *IBM Informix High-Performance Loader User's Guide*.
- Neprotokolující tabulky

  Databázový server poskytuje podporu pro:
  - Vytváření neprotokolujících nebo protokolujících tabulek v protokolující databázi.
  - Úprava tabulky z neprotokolující na protokolující a naopak.

Tyto dva typy tabulek jsou STANDARD (tabulka s protokolováním) a RAW (neprotokolující tabulka). K zavedení tabulek s přímým přístupem můžete použít libovolný nástroj, například **dbimport** nebo HPL.

Následující část popisuje:

- Výhody protokolujících a neprotokolujících tabulek
- Podrobné postupy zavedení dat pomocí protokolujících tabulek

Informace o obnově standardních tabulek a tabulek s přímým přístupem naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Výhody protokolujících tabulek

Typ STANDARD, který se vztahuje k tabulce v protokolující databázi předchozí verze, je výchozí. Při vyvolání příkazu CREATE TABLE bez zadání typu tabulky, vytvoříte standardní tabulku.

Standardní tabulky mají následující vlastnosti:

- Protokolování, které umožňuje odvolání a rychlé obnovení
- Obnovení ze zálohy
- Všechny operace vložení, odstranění a aktualizace
- Omezení k zachování integrity dat
- Indexy pro rychlé vyhledání malého počtu řádků

Aplikace OLTP obvykle používají standardní tabulky. Aplikace OLTP mají obvykle následující charakteristiku:

- Transakce vložení, aktualizace a odstranění v reálném čase

  Protokolování a obnova těchto transakcí je pro zachování dat rozhodující. Zamykání je rozhodující, aby byl umožněn souběžný přístup pro zajištění konzistence vybraných dat.
- Současná aktualizace, vkládání nebo odstranění jednoho nebo několika řádků

  Indexy k těmto řádkům zrychlují přístup. Index vyžaduje pro přístup k příslušnému řádku pouze méně operací vstupu - výstupu, ale prohledávání tabulky vyžaduje mnoho operací vstupu - výstupu.

Výhody neprotokolujících tabulek

Výhoda neprotokolujících tabulek je v tom, že můžete rychle zavést velmi velké tabulky datových skladů, protože mají následující charakteristiky:

- Nepoužívají k protokolování procesor ani prostředky vstupu - výstupu.
- Zabraňují problémům, jako je vyčerpání místa logického protokolu.
- Při rychlém zavedení jsou výlučně uzamčeny, aby během zavedení k tabulce nemohl přistupovat žádný jiný uživatel.
- Tabulky s přímým přístupem nepodporují referenční omezení a jedinečná omezení, takže je odstraněno zahlcení při kontrole omezení.

Rychlé zavedení velké existující standardní tabulky:

1. Vypusťte indexy, referenční omezení a jedinečná omezení.
2. Změňte tabulku na tabulku neprotokolující.

   Následující ukázka příkazu SQL mění tabulku STANDARD na tabulku neprotokolující:

   ```
   ALTER TABLE largetab TYPE(RAW);
   ```
3. Zaveďte tabulku pomocí obslužného programu pro zavedení, například **dbexport** nebo High-Performance Loader (zavaděč HPL).

Další informace o **dbexport** a **dbload** naleznete v příručce *IBM Informix Migration Guide*. Další informace o Zavaděč HPL naleznete v příručce *IBM Informix High-Performance Loader User's Guide*.

4. Proveďte zálohu úrovně 0 neprotokolující tabulky.

   U všech neprotokolujících tabulek, které byly změněny, musíte před jejich převodem na typ STANDARD vytvořit zálohu úrovně 0. Záloha úrovně 0 poskytuje bod spuštění, od kterého se obnovují data.

5. Před použitím v transakci změňte neprotokolující tabulku na tabulku protokolující.

   Následující ukázkový příkaz SQL změní tabulku s přímým přístupem na standardní tabulku:

   ```
   ALTER TABLE velká_tabulka TYPE(STANDARD);
   ```

   Upozornění: Není doporučeno používat neprotokolující tabulky v rámci transakce, ve které může měnit data vice uživatelů. Pokud potřebujete v transakci použít neprotokolující tabulku, nastavte úroveň izolace na opakovatelné čtení nebo zamkněte tabulku ve výlučném režimu, aby se zabránilo problémům se souběžností.

   Další informace o standardních tabulkách naleznete v předchozí části "Výhody protokolujících tabulek" na stránce 6-30.

6. Znovu vytvořte indexy, referenční omezení a jedinečná omezení.

Rychlé zavedení nové velké tabulky:

1. Vytvořte neprotokolující tabulku v protokolované databázi.

   Následující ukázkové příkazy SQL vytváří neprotokolující tabulku:

   ```
   CREATE DATABASE historie WITH LOG;
   CONNECT TO DATABASE historie;
   CREATE RAW TABLE historie (...
   );
   ```

2. Zaveďte tabulku pomocí obslužného programu pro zavedení, například **dbexport** nebo High-Performance Loader (zavaděč HPL).

   Další informace o **dbexport** a **dbload** naleznete v příručce *IBM Informix Migration Guide*. Další informace o Zavaděč HPL naleznete v příručce *IBM Informix High-Performance Loader User's Guide*.

3. Proveďte zálohu úrovně 0 neprotokolující tabulky.

   U všech neprotokolujících tabulek, které byly změněny, musíte před jejich převodem na typ STANDARD vytvořit zálohu úrovně 0. Záloha úrovně 0 poskytuje bod spuštění, od kterého se obnovují data.

4. Před použitím v transakci změňte neprotokolující tabulku na tabulku protokolující.

   Následující ukázkový příkaz SQL změní tabulku s přímým přístupem na standardní tabulku:

   ```
   ALTER TABLE velká_tabulka TYPE(STANDARD);
   ```

   Upozornění: Není doporučeno používat neprotokolující tabulky v rámci transakce, ve které může měnit data vice uživatelů. Pokud potřebujete v transakci použít neprotokolující tabulku, nastavte úroveň izolace na opakovatelné čtení nebo zamkněte tabulku ve výlučném režimu, aby se zabránilo problémům se souběžností.

   Další informace o standardních tabulkách naleznete v předchozí části "Výhody protokolujících tabulek" na stránce 6-30.

5. Vytvořte indexy na sloupcích, které jsou často používány ve filtrech dotazů.
6. V případě potřeby vytvořte referenční omezení a jedinečná omezení.

Vypuštění indexů z důvodu efektivity aktualizací tabulky

v některých aplikacích můžete většinu aktualizací tabulky omezit na jeden časový interval. Systém můžete nastavit, aby se všechny aktualizace prováděly v noci nebo v určené dny.

Pokud jsou aktualizace prováděny v dávce, můžete během aktualizací vypustit všechny nejedinečné indexy a pak je znovu vytvořit. Tato strategie může mít dva pozitivní efekty:

- Aktualizační program funguje výrazně rychleji, pokud nemusí během aktualizace tabulek aktualizovat indexy.
- Znovu vytvářené indexy jsou efektivnější.

Další informace o tom, kdy vypouštět indexy, naleznete v části "Vypouštění indexů" na stránce 7-10.

Zavedení tabulky, která nemá indexy:

1. Vypuštěním zrušte tabulku (pokud existuje).
2. Vytvořte tabulku, aniž byste určili jedinečná omezení.
3. Zaveďte do této tabulky všechny řádky.
4. Změňte tabulku tak, aby používala jedinečná omezení.
5. Vytvořte nejedinečné indexy.

Pokud nemůžete zaručit, že zavedená data vyhovují všem jedinečným indexům, musíte vytvořit jedinečné indexy před zavedením řádků. Ušetříte čas, pokud jsou řádky předávány ve správném pořadí alespoň pro jeden index. Máte-li možnost volby, zvolte řádek s největším klíčem. Tato strategie minimalizuje počet listových stránek, které musejí být přečteny a zapsány.

Připojení nebo odpojení fragmentů

Mnoho zákazníků používá k provádění operací typu datových skladů příkazy ALTER FRAGMENT ATTACH a DETACH. Pomocí příkazu ALTER FRAGMENT DETACH lze segment dat tabulky rychle odstranit. Podobně, příkaz ALTER FRAGMENT ATTACH poskytuje díky využití technologie fragmentace možnost, jak zavést do existující tabulky inkrementálně velká množství dat.

Další informace o způsobu, jak využít zlepšení výkonu pomocí možností ATTACH a DETACH příkazu ALTER FRAGMENT, naleznete v části "Zvyšování výkonu připojených a odpojených fragmentů" na stránce 9-17.

Úprava definice tabulky

Ke zpracování příkazů ALTER TABLE v SQL používá databázový server jeden z několika algoritmů:

- Pomalé změny
- Změny na místě
- Rychlé změny

Pomalé změny

Pokud databázový server používá ke zpracování příkazu ALTER TABLE algoritmus pomalých změn, může být tabulka pro ostatní uživatele dlouhou dobu nedostupná, protože databázový server:

- Zamyká v průběhu operace ALTER TABLE tabulku ve výlučném režimu.
- Vytváří kopii tabulky za účelem převodu tabulky na novou definici.
- Převádí během operace ALTER TABLE datové řádky.

- Může považovat příkaz ALTER TABLE za dlouhou transakci a při překročení prahové hodnoty LTXHWM ji ukončit.

Databázový server používá algoritmus pomalých změn, pokud příkaz ALTER TABLE provádí změny sloupců, které nemůže provádět na místě:

- Přidání nebo vypuštění sloupce vytvořeného pomocí klíčového slova ROWIDS
- Vypuštění sloupce datového typu TEXT nebo BYTE
- Úprava sloupce SMALLINT na SERIAL nebo SERIAL8 nebo převod sloupce INT na SERIAL nebo SERIAL8
- Úprava datového typu sloupce, aby některé hodnoty nebylo možné převést na nový datový typ

  Pokud například převádíte sloupec datového typu INTEGER na CHAR(n), používá databázový server algoritmus pomalých změn, pokud je hodnota n menší než 11. Typ INTEGER vyžaduje 10 znaků a znaménko minus pro nejnižší možné záporné hodnoty.
- Změna datového typu sloupce fragmentace takovým způsobem, že převod může způsobit přesun řádků do dalšího fragmentu
- Přidání, vypuštění nebo úprava libovolného sloupce, pokud tabulka obsahuje uživatelsky definované datové typy nebo inteligentní velké objekty

Pokud používáte příkaz ALTER TABLE k úpravě původní velikosti nebo specifikace rezervy sloupců VARCHAR nebo NVARCHAR, provádí databázový server tyto změny jako pomalé změny místo použití mechanismu změn na místě.

Změny na místě

Algoritmus změn na místě poskytuje ve srovnání s algoritmem pomalých změn následující výkonnostní výhody:

- Zvyšuje dostupnost tabulky

  Pokud operace ALTER TABLE používá algoritmus změn na místě, mohou ostatní uživatelé k tabulce přistupovat dříve, protože databázový server zamyká tabulku pouze na dobu potřebnou k aktualizaci definice tabulky a k opětovnému vytvoření indexů, které obsahují změněné sloupce.

  Toto zvýšení dostupnosti může zvýšit propustnost pro aplikační systémy, které vyžadují provoz 24 hodin denně, 7 dní v týdnu.

  Pokud databázový server používá algoritmus změn na místě, zamyká tabulku na kratší dobu, než algoritmus pomalých změn, protože databázový server:

  – Nevytváří kopie tabulky k převodu na novou definici
  – Nepřevádí datové řádky během operace ALTER TABLE
  – Pokud řádky aktualizujete nebo vkládáte postupně, mění fyzické sloupce na místě nejnovější definice po operaci změny. Databázový server převádí řádky umístěné na všech stránkách, které jste aktualizovali.
- Vyžaduje méně prostoru, než algoritmus pomalých změn

  Pokud operace ALTER TABLE používá algoritmus pomalých změn, vytváří databázový server kopii tabulky za účelem převodu tabulky na novou definici. Operace ALTER TABLE vyžaduje prostor nejméně dvakrát větší než původní tabulka plus velikost protokolu.

  Pokud operace ALTER TABLE používá algoritmus změn na místě, může dojít u velmi velkých tabulek ke značné úspoře místa.
- Zlepšuje propustnost systému během operace ALTER TABLE

  Databázový server nepotřebuje během operace změny na místě protokolovat žádné změny dat tabulky. Pokud se změny neprotokolují, existují následující výhody:

  – U velkých tabulek může být úspora prostoru protokolu významná.

– Operace změny není dlouhá transakce.

Kdy databázový server používá algoritmus změn na místě: Algoritmus změn na místě používá databázový server pro určité typy operací, které určíte v klauzulích ADD, DROP a MODIFY příkazu ALTER TABLE:

- Přidání sloupce nebo seznamu sloupců libovolného datového typu kromě sloupců přidaných pomocí klíčového slova ROWIDS
- Vypuštění sloupce libovolného datového typu kromě typů TEXT a BYTE a sloupců vytvořených pomocí klíčového slova ROWIDS
- Přidání nebo vypuštění sloupce vytvořeného pomocí klíčového slova CRCOLS
- Úprava sloupce, pro který může databázový server převést všechny možné hodnoty starého datového typu na nový datový typ
- Úprava sloupce, který je částí výrazu fragmentace při změně hodnoty, nevyžaduje po konverzi přesun řádku z jednoho fragmentu do jiného fragmentu

Důležité: Pokud tabulka obsahuje uživatelské datové typy, datový typ VARCHAR nebo inteligentní velké objekty, nepoužívá databázový server algoritmus změn na místě, i když upravovaný sloupec obsahuje vestavěný datový typ.

Tabulka 6-2 zobrazuje podmínky, za kterých příkaz ALTER TABLE MODIFY používá ke zpracování algoritmus změn na místě.

Tabulka 6-2. Operace a podmínky klauzule MODIFY, které používají algoritmus změn na místě

Operace na sloupci	Podmínka
Převod sloupce SMALLINT na sloupec INTEGER	Vše
Převod sloupce SMALLINT na sloupec INTEGER8	Vše
Převod sloupce SMALLINT na sloupec DEC(p2,s2)	p2-s2 >= 5
Převod sloupce SMALLINT na sloupec DEC(p2)	p2-s2 >= 5 OR nf
Převod sloupce SMALLINT na sloupec SMALLFLOAT	Vše
Převod sloupce SMALLINT na sloupec FLOAT	Vše
Převod sloupce SMALLINT na sloupec CHAR(n)	n >= 6 AND nf
Převod sloupce INT na sloupec INTEGER8	Vše
Převod sloupce INT na sloupec DEC(p2,s2)	p2-s2 >= 10
Převod sloupce INT na sloupec DEC(p2)	p2 >= 10 OR nf
Převod sloupce INT na sloupec SMALLFLOAT	nf
Převod sloupce INT na sloupec FLOAT	Vše
Převod sloupce INT na sloupec CHAR(n)	n >= 11 AND nf
Převod sloupce SERIAL na sloupec INTEGER8	Vše
Převod sloupce SERIAL na sloupec DEC(p2,s2)	p2-s2 >= 10
Převod sloupce SERIAL na sloupec DEC(p2)	p2 >= 10 OR nf
Převod sloupce SERIAL na sloupec SMALLFLOAT	nf
Převod sloupce SERIAL na sloupec FLOAT	Vše
Převod sloupce SERIAL na sloupec CHAR(n)	n >= 11 AND nf
Převod sloupce SERIAL na sloupec SERIAL	Vše
Převod sloupce SERIAL na sloupec SERIAL8	Vše
Převod sloupce DEC(p1,s1) na sloupec SMALLINT	p1-s1 < 5 AND (s1 == 0 OR nf)

Tabulka 6-2. Operace a podmínky klauzule MODIFY, které používají algoritmus změn na místě (pokračování)

Operace na sloupci	Podmínka
Převod sloupce DEC(p1,s1) na sloupec INTEGER	p1-s1 < 10 AND (s1 == 0 OR nf)
Převod sloupce DEC(p1,s1) na sloupec INTEGER8	p1-s1 < 20 AND (s1 == 0 OR nf)
Převod sloupce DEC(p1,s1) na sloupec SERIAL	p1-s1 < 10 AND (s1 == 0 OR nf)
Převod sloupce DEC(p1,s1) na sloupec SERIAL8	p1-s1 < 20 AND (s1 == 0 OR nf)
Převod sloupce DEC(p1,s1) na sloupec DEC(p2,s2)	p2-s2 >= p1-s1 AND (s2 >= s1 OR nf)
Převod sloupce DEC(p1,s1) na sloupec DEC(p2)	p2 >= p1 OR nf
Převod sloupce DEC(p1,s1) na sloupec SMALLFLOAT	nf
Převod sloupce DEC(p1,s1) na sloupec FLOAT	nf
Převod sloupce DEC(p1,s1) na sloupec CHAR(n)	n >= 8 AND nf
Převod sloupce DEC(p1) na sloupec DEC(p2)	p2 >= p1 OR nf
Převod sloupce DEC(p1) na sloupec SMALLFLOAT	nf
Převod sloupce DEC(p1) na sloupec FLOAT	nf
Převod sloupce DEC(p1) na sloupec CHAR(n)	n >= 8 AND nf
Převod sloupce SMALLFLOAT na sloupec DEC(p2)	nf
Převod sloupce SMALLFLOAT na sloupec FLOAT	nf
Převod sloupce SMALLFLOAT na sloupec CHAR(n)	n >= 8 AND nf
Převod sloupce FLOAT na sloupec DEC(p2)	nf
Převod sloupce FLOAT na sloupec SMALLFLOAT	nf
Převod sloupce FLOAT na sloupec CHAR(n)	n >= 8 AND nf
Převod sloupce CHAR(m) na sloupec CHAR(n)	n >= m OR (nf AND not ANSI mode)
Zvýšení délky sloupce CHARACTER	Není v režimu ANSI
Zvýšení délky sloupce DECIMAL nebo MONEY	Vše

Poznámky:

- Typ sloupce DEC(p) se vztahuje k databázím, které nejsou ANSI a ve kterých se s těmito typy pracuje jako s čísly s pohyblivou řádovou čárkou.
- V databázích ANSI je typ DEC(p) ve výchozím nastavení DEC(p,0) a používá stejný algoritmus jako DEC(p,s).
- Podmínka nf označuje, že databázový server používá algoritmus změn na místě, pokud není upravovaný sloupec součástí výrazu fragmentace tabulky.
- Podmínka Vše označuje, že databázový server používá algoritmus změn na místě ve všech případech příslušné operace se sloupcem.

Úvahy o výkonu příkazů DML: Při každém provedení příkazu ALTER TABLE, který používá algoritmus změn na místě, vytváří databázový server novou verzi struktury tabulky. Databázový server zachovává všechny verze definic tabulek. Databázový server znovu nastavuje struktury verze a struktury změn, dokud není celá tabulka převedena do konečného formátu nebo není provedena pomalá změna.

Pokud databázový server zjistí během provádění příkazů jazyka DML (INSERT, UPDATE, DELETE, SELECT) libovolnou stránku zastaralé verze, provede následující akce:

- U příkazů UPDATE převede databázový server celou datovou stránku nebo stránky do konečného formátu.
- U příkazů INSERT převede databázový server vkládaný řádek do konečného formátu a vloží jej do stránky, do které se nejlépe vejde. Databázový server převede existující řádky v této stránce do konečného formátu.
- U příkazů DELETE databázový server nepřevádí datové stránky do konečného formátu.
- U příkazů SELECT databázový server nepřevádí datové stránky do konečného formátu.

  Pokud dotaz přistupuje k řádkům, které dosud nejsou převedeny na novou definici tabulky, může dojít k nepatrnému snížení výkonu konkrétního dotazu, protože databázový server před vrácením převádí každý řádek.

Úvahy o výkonu příkazů DDL: Volba **oncheck -pT** *tablename* zobrazuje verze datových stránek pro nevyřízené operace změn na místě. Všechny změny na místě jsou nevyřízené, pokud ještě existují datové stránky se starou definicí.

Obrázek 6-14 zobrazuje část výstupu, který vytváří následující příkaz **oncheck** po provedení čtyř operací změn na místě v ukázkové tabulce **customer**:

```
oncheck -pT stores_demo:customer

...
Home Data Page Version Summary

            Version           Count

            0 (oldest)          2
            1                   0
            2                   0
            3                   0
            4 (current)         0
...
```

Obrázek 6-14. Vzorový výstup příkazu oncheck -pT pro tabulku customer

Ve sloupci **Count**, který znázorňuje Obrázek 6-14, je zobrazen počet stránek, které aktuálně používají tuto verzi definice tabulky. Tento výstup příkazu **oncheck** zobrazuje, že čtyři verze jsou nevyřešené:

- Nová hodnota 2 ve sloupci **Count** nejstarší verze označuje, že dvě stránky používají nejstarší verzi.
- Hodnota 0 ve sloupci **Count** dalších čtyř verzí označuje, že na nejnovější verzi tabulky nebyly převedeny žádné stránky.

Důležité: Pokud na tabulce provádíte více operací změn na místě, potřebuje každý následný příkaz ALTER TABLE k provedení delší čas, než předchozí příkazy. Proto je doporučeno nemít na tabulce více než přibližně 50 až 60 nevyřízených změn. Větší počet nevyřízených změn má vliv pouze na následné příkazy ALTER TABL, ale nesnižuje výkon příkazů SELECT.

Datové stránky můžete převést na nejnovější definici pomocí příkazu UPDATE. Následující příkaz, který nastavuje hodnotu sloupce na existující hodnotu, například způsobuje, že databázový server převádí datové stránky na nejnovější definici:

```
UPDATE tab1 SET col1 = col1;
```

Tento příkaz nemění hodnoty dat, ale převádí formát datových stránek na novou definici.

Po provedení aktualizace na všech stránkách tabulky zobrazí příkaz **oncheck -pT** ve sloupci **Count** celkový počet datových stránek aktuální verze tabulky.

Operace změn, které nepoužívají algoritmus změn na místě: Databázový server nepoužívá algoritmus změn na místě v následujících situacích:

- Pokud se používá více než jeden algoritmus

  Pokud příkaz ALTER TABLE obsahuje více než jednu změnu, používá databázový server při provedení příkazu algoritmus s nejnižším výkonem.

  Předpokládejme například, že příkaz ALTER TABLE MODIFY převádí sloupec SMALLINT na sloupec DEC(8,2) a převádí sloupec INTEGER na sloupec CHAR(8). Převod prvního sloupce je operace změny na místě, ale převod druhého sloupce je operace pomalé změny. Databázový server používá k provedení tohoto příkazu algoritmus pomalých změn.
- Pokud se mají hodnoty přesunout do jiného fragmentu

  Předpokládejme například, že máte tabulku se dvěma celočíselnými sloupci a následující výraz fragmentu:

  ```
  col1 < col2 v prostoru dbspace1, zbytek v prostoru dbspace2
  ```

  Pokud v následující části provedete příkaz ALTER TABLE MODIFY, uloží databázový server řádek (4, 30) v prostoru **dbspace1** před změnou, ale v prostoru **dbspace2** jej uloží po operaci změny, protože 4 < 30, ale “30” < “4”.

Úprava sloupce, který je částí indexu: Pokud jsou měněné sloupce částí indexu, mění se tabulka na místě, ale v tomto případě databázový server implicitně znovu vytvoří indexy. Pokud nepotřebujete znovu vytvořit index, můžete jej před provedením operace vypustit nebo zakázat. Provedením těchto kroků lze zlepšit výkon.

Pokud je upravovaný sloupec primárním klíčem nebo cizím klíčem a chcete toto omezení zachovat, musíte v příkazu ALTER TABLE tato klíčová slova znovu zadat a databázový server vytvoří index znovu.

Předpokládejme například, že vytváříte tabulky a upravujete nadřazenou tabulku pomocí následujících příkazů SQL:

```
CREATE TABLE parent
   (si smallint primary key constraint pkey);
CREATE TABLE child
   (si smallint references parent on delete cascade
   constraint ckey);
INSERT INTO parent (si) VALUES (1);
INSERT INTO parent (si) VALUES (2);
INSERT INTO child (si) VALUES (1);
INSERT INTO child (si) VALUES (2);
ALTER TABLE parent
     MODIFY (si int PRIMARY KEY constraint PKEY);
```

Tento příklad ALTER TABLE převádí sloupec SMALLINT na sloupec INT. Databázový server zachovává primární klíč, protože příkaz ALTER TABLE určuje klíčová slova PRIMARY KEY a omezení PKEY. Databázový server ale vypouští všechny referenční závislosti na tomto primárním klíči. Musíte proto také zadat následující přikaz ALTER TABLE pro podřízenou tabulku:

```
ALTER TABLE child
   MODIFY       (si int references parent on delete cascade
         constraint ckey);
```

I když operace ALTER TABLE na sloupci primárního klíče nebo cizího klíče znovu vytváří index, využívá databázový server stále výhodu algoritmu změn na místě. Algoritmus změn na místě poskytuje následující výkonnostní výhody:

- Nevytváří kopii tabulky za účelem převodu tabulky na novou definici.
- Nepřevádí datové řádky během operace změny.
- Nepřevádí na tabulce všechny indexy.

Upozornění: Pokud upravujete tabulku, která je částí pohledu, musíte pohled vytvořit znovu, abyste získali nejnovější definici tabulky.

Rychlá změna

Databázový server používá algoritmus rychlých změn, pokud příkaz ALTER TABLE mění atributy tabulky, ale nemá vliv na data: Databázový server používá algoritmus rychlých změn, pokud používáte příkaz ALTER TABLE k následujícím akcím:

- Změna velikosti další oblasti
- Přidání nebo vypuštění omezení
- Změna režimu uzamčení tabulky
- Změna atributu jedinečného indexu beze změny typu sloupce

S algoritmem rychlých změn databázový server drží zámek tabulky pouze krátkou dobu. V některých případech databázový server za účelem změny atributu zamyká pouze tabulky katalogu. V takovém případě je tabulka nedostupná pro dotazy pouze po krátkou dobu.

Zlepšení výkonu pomocí denormalizace datového modelu

Entitně-relační model dat, který popisuje příručka *IBM Informix Guide to SQL: Tutorial*, poskytuje tabulky, které neobsahují redundantní ani odvozená data. Podle principů teorie relačních databází jsou tyto tabulky dobře strukturované.

Někdy je nutné z důvodu splnění mimořádných požadavků na vysoký výkon upravit datový model takovými způsoby, které jsou z teoretického pohledu nežádoucí. Tato část popisuje některé úpravy a s nimi spojené náklady.

Zkrácení řádků

Tabulky s kratšími řádky obvykle umožňují vyšší výkon, než tabulky s delšími řádky, protože diskový vstup - výstup je prováděn po stránkách, nikoliv po řádcích. Čím kratší jsou řádky tabulky, tím více řádků se může nacházet na stránce. Čím více řádků se nachází na stránce, tím méně operací vstupu - výstupu je třeba provést při sekvenčním čtení tabulky a tím pravděpodobnější je, že nesekvenční přístup proběhne z vyrovnávací paměti.

Entitně-relační model dat umísťuje všechny atributy jedné entity do jedné tabulky pro tuto entitu. U některých entit tato strategie vytváří řádky nevhodné délky. Chcete-li zkrátit řádky, můžete sloupce rozdělit do oddělených tabulek, které jsou přidružené pomocí zdvojení klíčových hodnot v obou tabulkách. Protože jsou řádky kratší, dochází ke zlepšení výkonu dotazů.

Vyloučení dlouhých řádků

Nejobjemnější atributy jsou obvykle znakové řetězce. Chcete-li řádky zkrátit, můžete je odebrat z entitní tabulky. K vyloučení dlouhých řetězců můžete použít následující metody:

- Použití sloupců VARCHAR
- Použití dat typu TEXT
- Přesunutí řetězců do doprovodné tabulky
- Vytvoření tabulky symbolů

Použití sloupců VARCHAR

Globální podpora jazyků

Databáze může obsahovat sloupce CHAR, které můžete převést na sloupce VARCHAR. Sloupec VARCHAR můžete použít ke zkrácení průměrné délky řádku, pokud je průměrná délka textového řetězce ve sloupci CHAR alespoň o 2 bajty delší, než šířka sloupce. Další informace o jiných znakových datových typech naleznete v příručce *IBM Informix GLS User's Guide*.

Konec Globální podpora jazyků

Data typu VARCHAR jsou ihned kompatibilní s většinou existujících programů, formulářů a sestav. Možná bude třeba překompilovat všechny formuláře vytvořené pomocí nástrojů pro vývoj aplikací, aby rozpoznávaly sloupce VARCHAR. Formuláře a sestavy po úpravě schématu tabulek vždy otestujte na zkušební databázi.

Použití dat typu TEXT

Pokud řetězec vyplní polovinu diskové stránky nebo více, zvažte jeho převod na sloupec typu TEXT v odděleném prostoru blobspace. Sloupec ve stránce řádku má délku pouze 56 bajtů, což umožňuje umístit na stránku více řádků, než pokud řádek obsahuje dlouhý řetězec. Datový typ TEXT ale není automaticky kompatibilní s existujícími programy. Aplikace potřebná k načtení hodnoty typu TEXT je trochu složitější, než kód pro načtení hodnoty typu CHAR do programu.

Přesunutí řetězců do doprovodné tabulky

Řetězce, které jsou menší než polovina stránky, plýtvají místem na disku v případě, že s nimi pracujete jako s daty typu TEXT, ale můžete je přesunout z hlavní tabulky do doprovodné tabulky.

Vytvoření tabulky symbolů

Pokud sloupec obsahuje řetězce, které nejsou ve všech řádcích jedinečné, můžete tyto řetězce přesunout do tabulky, ve které budou uloženy pouze jedinečné kopie.

Sloupec **customer.city** například obsahuje názvy měst. Některé názvy měst se ve sloupci opakují a většina sloupců má v poli nějaká prázdná místa. Pomocí datového typu VARCHAR lze odstranit prázdná místa, ale nikoliv zdvojení.

Jak zobrazuje následující příklad, můžete vytvořit tabulku **cities**:

```
CREATE TABLE cities (
   city_num SERIAL PRIMARY KEY,
   city_name VARCHAR(40) UNIQUE
)
```

Definici tabulky **customer** můžete změnit tak, aby její sloupec **city** obsahoval cizí klíč, který závisí na sloupci **city_num** v tabulce **cities**.

Ke vložení města nového zákazníka do tabulky města **cities** musíte změnit všechny programy, které vkládají nový řádek do tabulky **customer.** Kód vrácený databázovým serverem v poli **SQLCODE** komunikační oblasti jazyka SQL (SQLCA) může indikovat, že vložení selhalo z důvodu zdvojeného klíče. Nejedná se o logickou chybu, jedná se pouze o to, že je se v tomto již nachází existující zákazník. Další informace o oblasti SQLCA naleznete v příručce *IBM Informix Guide to SQL: Tutorial*.

Kromě změny všech programů pro vkládání dat musíte změnit také všechny programy a uložené dotazy, které získávají název města. Programy a vložené dotazy musí propojení s novou tabulkou **cities**, aby z ní mohly získat data. Nárůst složitosti v programech

vkládajících řádky a v některých dotazech je výsledkem ustoupení od teoretické správnosti datového modelu. Před provedením změny se ujistěte, že změna přinese úsporu místa na disku nebo času provádění.

Rozdělení širokých tabulek

Zvažte vliv všech atributů entity, která má příliš široké řádky, na dobrý výkon. Najděte určitý způsob nebo princip, jak je rozdělit na dvě skupiny. Rozdělte tabulku na dvě tabulky, primární a doprovodnou tabulky, ve kterých se bude opakovat primární klíč. Kratší řádky umožní rychleji provádět dotazy nebo aktualizace jednotlivých tabulek.

Rozdělení podle objemu

Jeden z principů, podle kterého lze rozdělovat tabulku, je objem. Objemné atributy, obvykle znakové řetězce, přesuňte do doplňkové tabulky. V primární tabulce ponechte číselné a jiné malé atributy. V ukázkové databázi můžete rozdělit sloupec **ship_instruct** z tabulky **orders**. Doprovodnou tabulku můžete nazvat **orders_ship**. Má dva sloupce, primární klíč, který je kopií sloupce **orders.order_num** a původní sloupec **ship_instruct**.

Rozdělení podle četnosti použití

Jiným principem rozdělení na entity je četnost použití. Pokud se k některým atributům přistupuje zřídka, přesuňte je do doprovodné tabulky. V demonstrační databázi se například pouze jeden program dotazuje na sloupce **ship_instruct**, **ship_weight** a **ship_charge**. V tom případě je můžete přesunout do doprovodné tabulky.

Rozdělení podle četnosti aktualizace

Aktualizace trvají déle než dotazy a aktualizující programy během procesu aktualizace zamykají indexové tabulky a řádky dat, a tím zabraňují dotazujícím programům v přístupu k tabulkám. Pokud můžete jednu tabulku rozdělit na dvě doprovodné tabulky, jednu s častěji aktualizovanými entitami a druhou s častěji dotazovanými entitami, můžete často také zlepšit celkovou dobu odezvy.

Náklady na doprovodné tabulky

Rozdělení tabulky spotřebuje určité místo na disku a zvýší složitost. Pro každý řádek existují dvě kopie primárního klíče, v každé tabulce jedna. Existují také dva indexy primárního klíče. Počet přidaných stránek můžete odhadnout pomocí metod ve výše uvedených částech.

Musíte upravit existující programy, sestavy a formuláře, které používají výraz SELECT *, protože se vrací méně sloupců. Programy, sestavy a formuláře, které používají atributy z obou tabulek, musí spojit tabulky provedením operace spojení.

V tom případě jsou při vložení nebo odstranění tabulky upraveny dvě tabulky místo jedné. Pokud úpravu dvou tabulek nekoordinujete (například jejich provedením v rámci transakce), může dojít ke ztrátě sémantické integrity.

Redundantní data

Normalizované tabulky neobsahují redundantní data. Každý atribut se nachází pouze v jedné tabulce. Normalizované tabulky také neobsahují žádná odvozená data. Místo toho jsou data, která lze vypočítat z existujících atributů, vybrána jako výraz založený na těchto atributech.

Normalizace tabulek minimalizuje množství použitého místa na disku a co nejvíce zjednodušuje aktualizaci tabulek. Normalizované tabulky však mohou vynutit časté použití spojení a agregačních funkcí a tyto procesy mohou být časově náročné.

Jako alternativu můžete zavést nové sloupce, které obsahují redundantní data, za předpokladu, že chápete související omezení.

Přidávání redundantních dat

Správný datový model zabraňuje redundanci tím, že uchovává atributy pouze v tabulce entity, kterou popisuje. Pokud jsou data atributu třeba v jiném kontextu, vytváříte spojení tabulek. Spojení tabulek ale zabírá čas. Pokud mají častá spojení vliv na výkon, můžete je odstranit zdvojením spojených dat do jiné tabulky.

V databázi **stores_demo** obsahuje tabulka **manufact** názvy výrobců jejich dodací lhůty. Aktuální pracovní databáze může obsahovat mnoho dalších atributů dodavatele, například adresu a jméno obchodního zástupce.

Obsah tabulky **manufact** je zejména doplňkem tabulky **stock**. Předpokládejme časově kritickou aplikaci, která se často odkazuje na dodací lhůtu určitého produktu, ale ne na žádné jiné sloupce tabulky **manufact**. Kvůli každému takovému odkazu musí databázový server při provedení hledání přečíst dvě nebo tři stránky.

Do tabulky **stock** můžete přidat nový sloupec **lead_time** a vyplnit jej kopiemi sloupce **lead_time** odpovídajících řádků tabulky **manufact**. Tímto uspořádáním se odstraní hledání a aplikace se zrychlí.

Jako u odvozených dat zabírají redundantní data místo a představují riziko pro integritu. V příkladu popsaném v předchozím odstavci může existovat mnoho dalších kopií dodací lhůty pro jednotlivé výrobce. (Každý výrobce se může vícekrát vyskytovat v tabulce **stock**.) Programy, které vkládají nebo aktualizují řádky v tabulce **manufact** musí také aktualizovat více řádků tabulky **stock**.

Riziko pro integritu spočívá v tom, že redundantní kopie nemusí být správné. Pokud se v tabulce **manufact** změní dodací lhůta, je sloupec **stock** neaktuální do jeho příští aktualizace. Stejně jako u odvozených dat definujte podmínky, za kterých mohou být chybná redundantní data.

Další informace o návrhu databáze naleznete v příručce *IBM Informix Database Design and Implementation Guide*.

Snížení prostoru na disku pomocí dalších řádků na stránku v tabulkách s proměnnou délkou řádků

Nastavením konfiguračního parametru MAX_FILL_DATA_PAGES na hodnotu 1 můžete povolit, aby databázový server do tabulek s proměnnou délkou řádků vložil více řádků na stránku.

Možnost ukládat na stránku více řádků různé délky poskytuje následující potenciální výhody:

- Omezení diskového prostoru požadovaného k uložení dat
- Povolení serveru používat efektivněji společnou oblast vyrovnávací paměti
- Omezení času prohledávání tabulek

Možnost použití parametru MAX_FILL_DATA_PAGES, který umožňuje ukládat na stránku více řádků různé délky, má následující potenciální nevýhody:

- Server může ukládat řádky v různém fyzickém pořadí.
- Během zaplňování řádků mohou aktualizace provedené ve sloupcích s proměnnou délkou způsobit rozšíření řádku, takže se již nevejde celý na stránku. To způsobí, že server rozdělí řádek na dvě stránky a zvýší se přístupová doba k řádku.

Pokud je povolen konfigurační parametr MAX_FILL_DATA_PAGES, přidá server nový řádek do nedávno upravené stránky s existujícími řádky, pokud přidání řádku ponechá

alespoň 10 procent stránky volné k budoucímu rozšíření všech řádků ve stránce. Pokud konfigurační parametr MAX_FILL_DATA_PAGES není povolen, přidá server řádek, pouze pokud je na stránce dostatek místa, aby se mohly řádky zvětšit na maximální délku.

Pokud povolíte konfigurační parametr MAX_FILL_DATA_PAGES a chcete ovlivnit existující řádky proměnné délky, musí být existující tabulky znovu zavedeny.

Povolení serveru ukládat řádky na stránku

Chcete-li povolit serveru vkládání dalších řádků na stránku do tabulek s proměnnou délkou, nastavte parametr MAX_FILL_DATA_PAGES na hodnotu 1.

Kapitola 7. Úvahy o výkonu indexů

Obsah kapitoly

Tato kapitola popisuje úvahy o výkonu spojené s indexy. Zabývá se posouzením volného místa, volby indexů, které mají být vytvořeny, správy indexů a vytváření a vypouštění indexů v online prostředí.

Odhad indexových stránek

Indexové stránky přidružené k tabulce mohou významně zvětšit velikost prostoru dbspace. Databázový server vytváří index při výchozím nastavení ve stejném prostoru dbspace, jako je tabulka, ale v jiném prostoru tblspace, než je tabulka. Chcete-li index umístit do odděleného prostoru dbspace, zadejte v příkazu CREATE INDEX klíčové slovo IN. Informace

o prostorech dbspace a o tom, které objekty jsou ukládány do prostoru tblspace naleznete v kapitole o ukládání dat v příručce *IBM Informix Administrator's Guide*.

I když velikost oblasti indexu nelze explicitně určit, můžete odhadnout počet stránek, které by index mohl zabrat, a určit tak, zda je prostorům dbspace přidělen dostatek místa.

Velikosti oblasti indexů

Databázový server určuje velikost oblasti indexu na základě velikosti oblasti odpovídající tabulky bez ohledu na to, zda je index fragmentovaný nebo ne.

Odhad velikosti oblasti připojeného indexu

Databázový server přiřazuje vhodnou velikost oblasti pro připojený index pomocí poměru velikosti klíče indexu a velikosti řádku, jak ukazuje následující vzorec:

```
Velikost oblasti indexu = (index_key_size /
table_row_size) *
   table_extent_size
```

index_key_size je celková šířka indexovaných sloupců plus 4 pro deskriptor klíče.

table_row_size je součet všech sloupců v daném řádku.

table_extent_size
je hodnota, kterou zadáváte v klíčovém slově EXTENT SIZE příkazu CREATE TABLE.

Databázový server používá tento poměr také pro velikost další oblasti indexu:

```
velikost další oblasti indexu =
(index_key_size/table_row_size)*
   table_next_extent_size
```

Odhad velikosti oblasti odpojeného indexu

Databázový server přiřazuje vhodnou velikost oblasti pro odpojený index pomocí poměru velikosti klíče indexu plus nějaké bajty režie a velikosti řádku, jak ukazuje následující vzorec:

```
Velikost oblasti odpojeného indexu = ( (index_key_size +
9) /                              table_row_size) *
 table_extent_size
```

Předpokládejme, že máme následující hodnoty:

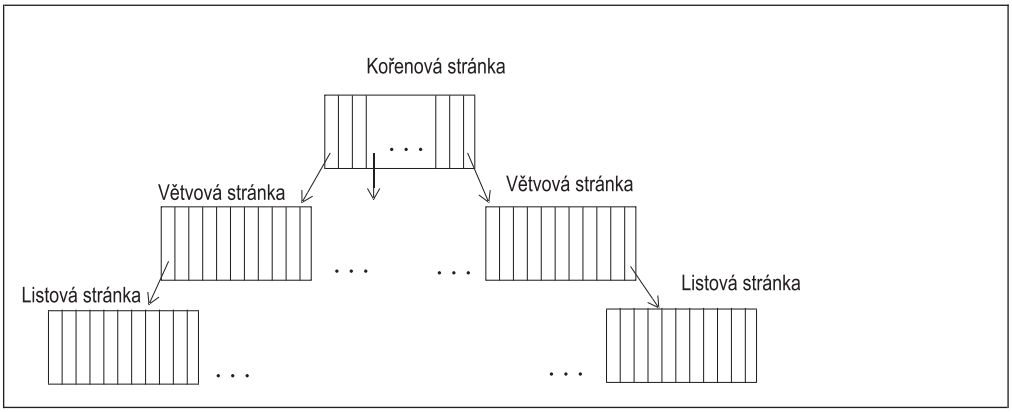
```
index_key_size = 8 bajtů
table_row_size = 33 bajtů
table_extent_size = 150 * 2kilobajtová stránka
```

Výše uvedený vzorec vypočítá velikost oblasti takto:

```
Velikost oblasti odpojeného indexu = ( (8 + 9)/
33) * 150 * 2kilobajtová stránka
                          = (17/33) * 300 kB
                          = 154 kB
```

Odhad konvenčních indexových stránek

Obrázek 7-1 zobrazuje uspořádání indexu jako hierarchie stránek (technicky se jedná o *B-strom*). Nejvyšší úroveň hierarchie obsahuje jednu *kořenovou stránku*. Střední úrovně obsahují v případě potřeby *stránky - větve*. Každá větev obsahuje položky, které vidí podmnožinu stránek v další úrovni indexu. Nejnižší úroveň indexu obsahuje množinu *stránek - listů*. Každý list obsahuje seznam položek indexu, které vidí řádky v tabulce.



Obrázek 7-1. Struktura B-stromu indexu

Počet úrovní potřebných k uchování indexu závisí na počtu jedinečných klíčů v indexu a na počtu položek indexu, který může každá stránka uchovávat. Počet položek na stránku závisí na velikosti sloupců, které jsou právě indexovány.

Pokud indexová stránka pro danou tabulku může uchovávat 100 klíčů, vyžaduje tabulka se 100 řádky jedinou úroveň indexu: kořenovou stránku. Jakmile tato tabulka překročí 100 řádků a dosáhne velikosti od 101 do 10000 řádků, bude vyžadovat index se dvěma úrovněmi: kořenovou stránku a 2 až 100 listových stránek. Jakmile tato tabulka překročí 10000 řádků a dosáhne velikosti mezi 10001 a 1000000 řádků, bude vyžadovat tři úrovně indexu: kořenovou stránku, sadu 100 stránek - větví a sadu až 10000 listových stránek.

Položky indexu obsažené v listových stránkách jsou seřazeny podle hodnot klíčů. Položka indexu se skládá z *klíče* a jednoho nebo více *ukazatelů řádku*. Klíč je kopií indexovaných sloupců jednoho řádku dat. Ukazatel řádku poskytuje adresu, pomocí které lze řádek obsahující klíč vyhledat. Jedinečný index obsahuje jednu položku indexu pro každý řádek tabulky.

Informace o speciálních indexech pro systém Dynamic Server naleznete v části “Indexy v uživatelských datových typech” na stránce 7-15.

Odhad počtu indexových stránek:

1. Sečtěte celkovou šířku indexovaných sloupců.

   Tato hodnota je dále nazývána colsize. Přičtete-li k hodnotě *colsize* číslo 4, dostanete hodnotu *keysize*, skutečnou velikost klíče v indexu.

   Má-li například *colsize* hodnotu 6, má *keysize* hodnotu 10.
2. Vypočtěte očekávaný podíl jedinečných položek a celkového počtu řádků.

   Ve vzorcích v následujících krocích je tato hodnota označena jako *propunique*.

   Je-li index jedinečný nebo má jen málo duplicitních hodnot, použijte pro *propunique* hodnotu 1.

   Pokud duplicitní položky představují významnou část položek, získáte hodnotu *propunique* vyjádřenou zlomkem jako podíl počtu jedinečných položek indexu a počtu řádků v tabulce. Pokud je například počet řádků v tabulce 4 000 000 a počet jedinečných položek indexu je 1 000 000, má *propunique* hodnotu 0,25.

   Je-li výsledná hodnota pro *propunique* menší než 0,01, používejte v následujících výpočtech hodnotu 0,01.
3. Odhadněte velikost běžné položky indexu pomocí jednoho z následujících vzorců v závislosti na tom, zda je tabulka fragmentovaná, nebo ne:

   a. Pro nefragmentované tabulky použijte následující vzorec:

```
entrysize = (keysize * propunique) + 5 + 4
```

Hodnota 5 představuje počet bajtů pro ukazatel řádku v nefragmentované tabulce.

Pro indexy, které nejsou jedinečné, ukládá databázový server ukazatel každého řádku do uzlu indexu, ale hodnotu klíče uloží pouze jednou. Hodnota entrysize představuje průměrnou délku každé položky indexu, i když některé položky se skládají pouze z ukazatele řádku. Má-li například *propunique* hodnotu 0,25, průměrný počet řádků pro každou hodnotu jedinečného klíče je 4. Má-li *keysize* hodnotu 10, má *entrysize* hodnotu 7,5. Následující výpočet udává požadované místo pro všechny čtyři řádky:

```
místo pro čtyři řádky = 4 * 7,5 = 30
```

Tento požadavek na místo je stejný, počítáte-li ho pro hodnotu klíče a přičtete čtyři ukazatele řádku podle ná sledujícího vzorce:

```
místo pro čtyři řádky = 10 + (4 * 5) = 30
```

b. Pro fragmentované tabulky použijte následující vzorec:

```
entrysize = (keysize * propunique) + 9 + 4
```

Hodnota 9 představuje počet bajtů pro ukazatel řádku ve fragmentované tabulce.

4. Odhadněte počet položek na jednu indexovou stránku pomocí následujícího vzorce:

```
pagents = trunc(pagefree/entrysize)
```

pagefree je velikost stránky minus hlavička stránky (2020 pro stránku o velikosti 2 kilobajty).

entrysize je hodnota, kterou odhadujete v kroku 3.

Zápis funkce **trunc()** označuje, že byste měli zaokrouhlit dolů na nejbližší celočíselnou hodnotu.

5. Odhadněte počet stránek - listů pomocí následujícího vzorce:

```
leaves = ceiling(rows/pagents)
```

rows je očekávaný počet řádků v tabulce.

pagents je hodnota, kterou odhadujete v kroku 4.

Zápis funkce **ceiling()** označuje, že byste měli zaokrouhlit nahoru na nejbližší celočíselnou hodnotu.

6. Odhadněte počet větví na druhé úrovni indexu pomocí následujícího vzorce:

$branches_0$ = ceiling(*leaves*/*node_ents*)

Vypočítejte hodnotu pro *node_ents* pomocí následujícího vzorce:

```
node_ents = trunc( pagefree / ( keysize + 4) + 4)
```

pagefree je stejná hodnota jako v kroku 4.

keysize je hodnota *colsize* plus 4 získaná v kroku 1.

V tomto vzorci představuje číslo 4 počet bajtů pro ukazatel listového uzlu.

7. Má-li položka $branches_0$ větší hodnotu než 1, zůstane v indexu více úrovní.

Počet stránek obsažených v další úrovni indexu můžete vypočítat pomocí následujícího vzorce:

$branches_{n+1}$ = ceiling(*$branches_n$*/*node_ents*)

$branches_n$ je počet větví pro poslední vypočítanou úroveň indexu.

$branches_{n+1}$ je počet větví v další úrovni.

node_ents je hodnota, kterou jste odhadli v kroku 6.

8. Opakujte výpočet v kroku 7 pro každou úroveň indexu, dokud nebude hodnota $branches_{n+1}$ rovna 1.

9. Sečtěte celkový počet stránek pro všechny úrovně větví vypočítané v krocích 6 až 8. Tento součet se nazývá *branchtotal*.
10. Počet stránek v kompaktním indexu můžete vypočítat pomocí následujícího vzorce:

    ```
    compactpages = (leaves + branchtotal)
    ```

11. Používá-li vaše instance databázového serveru pro indexy faktor výplně, velikost indexu se zvětší.

    Výchozí hodnotou faktoru výplně je 90 procent. Hodnotu faktoru výplně pro všechny indexy můžete změnit pomocí konfiguračního parametru FILLFACTOR. Můžete také změnit hodnoty faktoru výplně pro jednotlivé indexy, a to pomocí klauzule FILLFACTOR příkazu CREATE INDEX v jazyce SQL.

    Faktor výplně můžete do odhadu pro indexové stránky začlenit pomocí následujícího vzorce:

    ```
    indexpages = 100 * compactpages / FILLFACTOR
    ```

Předchozí odhad je pouze vodítko. Počet položek indexu na stránce se může měnit v závislosti na tom, jak jsou stávající řádky odstraňovány a nové vkládány. Tento způsob odhadování indexových stránek dává pro většinu indexů umírněný (vysoký) odhad. Chcete-li získat přesnější odhad, vytvořte rozsáhlý testovací index se skutečnými daty a zjistěte jeho velikost pomocí obslužného programu **oncheck**.

Odhad indexových stránek pro prostorová a uživatelská data

Databázový server používá následující typy indexů:

- B-strom
- R-strom
- Indexy, které poskytuje modul DataBlade

Indexy B-stromu

Dynamic Server používá index B-stromu pro tyto hodnoty:

- Sloupce, které obsahují vestavěné datové typy (nazývané *tradiční index B-stromu*)

  Mezi vestavěné datové typy patří znakový typ, datum a čas, celé číslo, pohyblivá řádová čárka atd. Další informace o vestavěných datových typech naleznete v příručce *IBM Informix Guide to SQL: Reference*.
- Sloupce, které obsahují jednorozměrné uživatelské datové typy (nazývané *obecný index B-stromu*)

  Mezi uživatelské datové typy patří netransparentní a odlišené datové typy. Další informace o uživatelských datových typech naleznete v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.
- Hodnoty, které uživatelská funkce vrací (nazývané *funkční index*)

  Vrácená hodnota může mít vestavěný nebo uživatelský datový typ, ale nesmí být jednoduchý velký objekt (datový typ TEXT nebo BYTE) ani inteligentní velký objekt (datový typ BLOB nebo CLOB). Další informace o použití funkčních indexů naleznete v části “Použití funkčního indexu” na stránce 7-18.

Informace o tom, jak odhadovat velikost indexu B-stromu, naleznete v části “Odhad indexových stránek” na stránce 7-1.

Indexy R-stromu

Dynamic Server používá pro prostorová data (dvojrozměrná, trojrozměrná atd.) index R-stromu. Informace o určení velikosti indexu R-stromu naleznete v příručce *IBM Informix R-Tree Index User's Guide*.

Indexy, které poskytují moduly DataBlade

V systému Dynamic Server mají uživatelé přístup k uživatelským typům, které poskytují moduly DataBlade. Modul DataBlade může také poskytovat uživatelský index pro nový datový typ. Například modul Excalibur Text Search Datablade poskytuje index pro vyhledávání textových dat. Další informace naleznete v příručce *Excalibur Text Search DataBlade Module User's Guide*.

Další informace o typech dat a funkcí, které každý modul DataBlade poskytuje, naleznete v uživatelské příručce příslušného modulu DataBlade. Další informace o určování typů indexů dostupných ve vaší databázi naleznete v části “Určování dostupných přístupových metod” na stránce 7-17.

Správa indexů

Index je nezbytný v každém sloupci nebo kombinaci sloupců, které musejí být jedinečné. Přítomnost indexu také umožňuje optimalizátoru dotazů urychlit dotaz, jak popisuje část Kapitola 10, “Dotazy a optimalizátor dotazů”, na stránce 10-1. Optimalizátor může index používat následujícími způsoby:

- Nahradit opakovaná sekvenční prohledávání tabulky nesekvenčním přístupem
- Zabránit čtení dat řádků při zpracování výrazů, které obsahují pouze indexované sloupce
- Zabránit řazení (včetně vytvoření dočasné tabulky) při provádění klauzulí GROUP BY a ORDER BY

Z toho vyplývá, že index vhodného sloupce může během dotazu uložit tisíce, desítky tisíců, nebo ve výjimečných případech i miliony diskových operací. Ale indexy také znamenají rizika.

Prosorová rizika indexů

Prvním rizikem indexu je prostor na disku. Přítomnost indexu může k prostoru dbspace přidat mnoho stránek; je snadné mít tolik indexových stránek, kolik je stránek řádků v indexované tabulce. Také v prostředí, ve kterém je používáno několik jazyků, vyžadují indexy vytvořené pro jednotlivé jazyky další prostor na disku. Metoda odhadu je uvedena v části “Odhad indexových stránek” na stránce 7-1.

Při posuzování prostorového rizika zvažte také, zda je zvýšení velikosti stránky standardního nebo dočasného prostoru dbspace vhodné pro používané prostředí. Chcete-li vyšší délku klíče, než jaká je dostupná pro výchozí velikost stránky, můžete zvýšit velikost stránky. Zvýšíte-li velikost stránky, musí být tato velikost celočíselným násobkem výchozí velikosti stránky a nesmí překročit 16 kB.

Pravděpodobně nebudete chtít zvýšit velikost stránky, pokud vaše aplikace obsahuje řádky malé velikosti. Zvýšení velikosti stránky pro aplikaci, která náhodně přistupuje k malým řádkům, by mohlo snížit výkon. Zámek větší stránky také zamkne více řádků, a sníží tak v některých situacích souběžnost.

Informace o tom, jak lze určit velikost stránky pro standardní a dočasný prostor dbspace, naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Časová rizika indexů

Dalším rizikem je vždy čas, kdy je tabulka měněna. Následující popisy předpokládají, že vyhledání položky indexu vyžaduje, aby byly přečteny přibližně dvě stránky. Jedná se o případ, kdy se index skládá z kořenové stránky, jedné úrovně větví a množiny listových stránek. Předpokládáme, že kořenová stránka je již ve vyrovnávací paměti. Index pro velmi

rozsáhlou tabulku má alespoň dvě střední úrovně, je tedy nutné přečíst přibližně tři stránky, když databázový server odkazuje na takový index.

Předpokládejme, že jeden index se používá při hledání řádku, který je právě měněn. Stránky pro tento index by mohly být nalezeny ve vyrovnávacích pamětech stránek ve sdílené paměti pro databázový server. Stránky pro libovolné jiné indexy, které potřebují změnit, musejí být čteny z disku.

Za těchto předpokladů přidá údržba indexů čas různým druhům změn, jak ukazuje následující seznam:

- Odstraníte-li řádek z tabulky, musí databázový server odstranit položky tohoto řádku ze všech indexů.

  Databázový server musí vyhledat položku pro odstraněný řádek (dvě nebo tři stránky) a přepsat listovou stránku. Operace zápisu, která aktualizuje index, se provádí v paměti a listová stránka je vyprázdněna, jakmile je vyčištěna vyrovnávací paměť naposledy použitých stránek (LRU), která obsahuje změněnou stránku. Tato operace vyžaduje přístup ke dvěma nebo třem stránkám, aby mohly být v případě potřeby načteny indexové stránky, a přístup k jedné odložené stránce, aby mohla být změněná stránka zapsána.
- Vložíte-li řádek, musí databázový server vložit jeho položky do všech indexů.

  Databázový server musí v každém indexu najít místo, do kterého má vložit řádek (zapíše dvě nebo tři stránky), a místo, které má přepsat (jednu odloženou stránku). Jedná se celkem o tři nebo čtyři přístupy na stránku na jeden index.
- Aktualizujete-li řádek, musí databázový server vyhledat jeho položky v každém indexu, který se vztahuje ke změněnému sloupci (zapíše dvě nebo tři stránky).

  Databázový server musí přepsat listovou stránku, aby odstranil starou položku (odstraní jednu odloženou stránku), a potom ve stejném indexu vyhledá hodnotu nového sloupce (zapíše dvě nebo tři stránky) a vložený řádek (odstraní ještě jednu odloženou stránku).

Vkládání a odstraňování mění počet položek na listové stránce. I když prakticky každá operace *pagents* vyžaduje nějakou další práci se stránkou - listem, ať má být vyplněna, nebo vyprázdněna, tato práce zabírá méně než 1 procento času, je-li hodnota *pagents* větší než 100. Můžete ji často zanedbat, když odhadujete vliv vstupu - výstupu.

Stručně řečeno: Při náhodném vložení nebo odstranění řádku povolte tři až čtyři operace vstupu - výstupu na přidaných stránkách na jeden index. Při aktualizaci řádku povolte šest až osm operací vstupu - výstupu na stránkách pro každý index, který se vztahuje ke změněnému sloupci. Je-li transakce odvolána, musí být veškerá tato práce vrácena zpět. Proto může odvolání transakce trvat dlouho.

Protože samotná změna řádku vyžaduje jen dvě operace vstupu - výstupu se stránkami, je údržba indexu nepochybně časově nejnáročnější částí modifikace dat. Informace o jednom způsobu snížení tohoto rizika naleznete v části “Klastrování” na stránce 7-9.

Odstranění nevyžádaného indexového prostoru

Jednotkový proces na pozadí, tzv. prohledávání B-stromu, identifikuje index, který má nejvíce nevyžádaného indexového prostoru. Nevyžádaný indexový prostor snižuje výkon a je příčinou mimořádné práce serveru. Jakmile je zvolen index pro prohledávání, jsou v celém listu indexu vyhledány odstraněné (neaktualizované) položky, které byly potvrzeny, ale nebyly ještě z indexu odstraněny. Prohledávání B-stromu tyto položky v případě potřeby odstraní. Prohledávání B-stromu umožňuje dovoluje několik jednotkových procesů.

Počet jednotkových procesů prohledávání B-stromu, které mají být spuštěny, a prioritu jednotkových procesů prohledávání B-stromu při spuštění databázového serveru můžete určit pomocí konfiguračního parametru BTSCANNER. Podrobnější informace najdete v příručce *IBM Informix Administrator's Reference*.

Prohledávání B-stromu můžete vyvolat z příkazového řádku.

Výběr sloupců pro indexy

Indexy jsou vyžadovány pro sloupce, které musejí být jedinečné a které nejsou určeny jako primární klíče. Index přidejte také ke sloupcům, které splňují následující podmínky:

- Jsou použity ve spojeních, která nejsou určena jako cizí klíče.
- Jsou často používány ve výrazech pro filtry.
- Jsou často používány pro řazení a seskupování.
- Nezahrnují duplicitní klíče.
- Umožňují klastrované indexování.

Filtrované sloupce v rozsáhlých tabulkách

Je-li sloupec často používán při filtrování řádků rozsáhlé tabulky, zvažte přidání indexu k tomuto sloupci. Optimalizátor pak může požadované sloupce vybrat pomocí indexu, a zabránit tak sekvenčnímu prohledávání celé tabulky. Příkladem je tabulka, která obsahuje rozsáhlý seznam poštovních adres. Zjistíte-li, že sloupec s poštovním směrovacím číslem je často používán při filtrování podmnožiny řádků, měli byste uvažovat o přidání indexu k tomuto sloupci.

Tato strategie poskytuje časové úspory v síti pouze tehdy, je-li selektivita sloupce vysoká; tj. pokud pouze malá část řádků uchovává libovolnou z indexovaných hodnot. Nesekvenční přístup prostřednictvím indexu využívá více diskových operací vstupu - výstupu než sekvenční přístup, takže předává-li filtrovací výraz více než jednu čtvrtinu řádků, mohl by databázový server stejně číst tabulku sekvenčně.

Indexování sloupce používaného při filtrování šetří obvykle čas v následujících případech:

- Je-li sloupec používán ve filtrovacích výrazech v mnoha dotazech nebo v pomalých dotazech.
- Obsahuje-li sloupec nejméně 100 jedinečných hodnot.
- Je-li většina hodnot sloupce obsažena v méně než 10 procentech řádků.

Sloupce, podle kterých se má řadit a seskupovat

V případě, že je nutné seřadit nebo seskupit velké množství řádků, musí databázový server řádky uspořádat. Jedním způsobem, kterým může databázový server tuto úlohu provést, je vybrat všechny řádky do dočasné tabulky a tuto tabulku seřadit. Ale, jak je popsáno v části Kapitola 10, “Dotazy a optimalizátor dotazů”, na stránce 10-1, pokud jsou sloupce, podle kterých má být tabulka seřazena, indexované, načte někdy optimalizátor řádky pomocí indexu v seřazeném pořadí, a vyhne se tak závěrečnému řazení.

Protože klíče jsou v indexu v seřazeném pořadí, představuje index skutečně výsledek řazení tabulky. Přidáním indexu k sloupcům, podle kterých má být tabulka seřazena, můžete nahradit mnohá řazení během dotazování jediným řazením, pokud je vytvořen index.

Obcházení sloupců s duplicitními klíči

Pokud jsou v indexu povoleny duplicitní klíče, jsou položky, které odpovídají danému klíči, seskupeny v seznamech. Databázový server vyhledá pomocí těchto seznamů řádky, které odpovídají požadované hodnotě klíče. Je-li selektivita indexového sloupce vysoká, jsou tyto seznamy obecně krátké. Ale pokud se vyskytuje jen málo jedinečných hodnot, jsou tyto seznamy příliš dlouhé a mohou přecházet na několik listových stránek.

Přidání indexu ke sloupci, který má nízkou selektivitu (tj. malý počet odlišných hodnot relativně k počtu řádků), může snížit výkon. V takových případech musí databázový server

vyhledat celou množinu řádků, které odpovídají hodnotě klíče, a také zamknout všechna dotčená data a indexové stránky. Tento proces může také snižovat výkon ostatních požadavků na aktualizaci.

Chcete-li tento problém opravit, nahraďte index sloupce s nízkou selektivitou složeným indexem, který má vyšší selektivitu. Sloupec s nízkou selektivitou použijte jako hlavní sloupec a sloupec s vysokou selektivitou jako druhý sloupec v indexu. Složený index omezuje počet řádků, které musí databázový server prohledávát při hledání a uplatnění aktualizace.

Hodnoty klíče můžete rozšířit do libovolného jiného sloupce, pokud se jeho hodnota nemění, nebo se mění současně se skutečným klíčem. Čím je druhý sloupec kratší, tím lépe, protože jeho hodnoty se kopírují do indexu a zvyšují jeho velikost.

Klastrování

Klastrování je takový způsob uspořádání řádků tabulky, ve kterém jejich fyzické pořadí na disku odpovídá pořadí položek v indexu. (Nezaměňujte klastrovaný index s *optickým klastrem*, metodou pro společné ukládání logicky souvisejících dat typu TEXT nebo BYTE na optický svazek.)

Pokud víte, že je tabulka seřazena podle určitého indexu, můžete se řazení vyhnout. Máte také jistotu, že při vyhledávání v tomto sloupci tabulky je tabulka čtena v sekvenčním pořadí místo nesekvenčně. Tyto otázky jsou popisuje Kapitola 10, “Dotazy a optimalizátor dotazů”, na stránce 10-1.

Tip: Informace o vyloučení prokládaných oblastí změnou indexu na klastr naleznete v části “Vytvoření nebo úprava indexu v klastru” na stránce 6-27.

V databázi **stores_demo** má tabulka **orders** index **zip_ix** ve sloupci s poštovním směrovacím číslem. Následující příkaz způsobí, že databázový server seřadí tabulku **customer** sestupně podle poštovního směrovacího čísla:

```
ALTER INDEX zip_ix TO CLUSTER
```

Chcete-li klastrovat tabulku v neindexovaném sloupci, musíte vytvořit index. Následující příkaz změní pořadí tabulky **orders** podle data objednávky:

```
CREATE CLUSTERED INDEX o_date_ix ON orders (order_date ASC)
```

Chcete-li změnit pořadí tabulky, musí databázový server tabulku nejprve zkopírovat. V předchozím příkladu načte databázový server všechny řádky v tabulce a sestrojí index. Potom sekvenčně přečte položky indexu. Pro každou položku přečte odpovídající řádek a zkopíruje ho do nové tabulky. Řádky nové tabulky jsou v požadovaném pořadí. Tato nová tabulka nahradí starou tabulku.

Klastrování se nezachová, změníte-li tabulku. Vložíte-li nové řádky, uloží se fyzicky na konec tabulky bez ohledu na jejich obsah. Po aktualizaci řádků a změně hodnoty klastrujícího sloupce jsou řádky zapsány zpět na své původní místo v tabulce.

Klastrování lze obnovit, až bude pořadí řádků porušeno právě probíhajícími aktualizacemi. Následující příkaz mění pořadí tabulky tak, že obnovuje datové řádky v pořadí indexu:

```
ALTER INDEX o_date_ix TO CLUSTER
```

Nové klastrování je obvykle rychlejší než původní klastrování, protože načítání řádků tabulky, která je téměř klastrovaná, je podobné ve vlivu vstupu - výstupu na sekvenční prohledávání.

Klastrování a opětovné klastrování zabírá mnoho času a prostoru. Chcete-li se některému klastrování vyhnout, vytvářejte tabulku od začátku v požadovaném pořadí.

Vypouštění indexů

V některých aplikacích lze většinu tabulek omezit na jedno časové období. Pravděpodobně můžete systém nastavit tak, aby byly všechny aktualizace prováděny přes noc nebo v určených dnech.

Pokud jsou aktualizace prováděny v dávce, můžete během aktualizací vypustit všechny nejedinečné indexy a pak je znovu vytvořit. Tato strategie může mít následující pozitivní účinky:

- Aktualizační program může probíhat rychleji, má-li být aktualizováno méně indexů. Často lze vypuštění indexů, aktualizaci bez nich a jejich opětovné vytvoření provést za kratší dobu, než jakou trvá aktualizace s indexy. (Diskuse o časových rizicích aktualizace indexů je uvedena v části "Časová rizika indexů" na stránce 7-6.)
- Nově vytvořené indexy jsou účinnější. Časté aktualizace způsobují prodloužení struktury indexu a index pak obsahuje mnoho částečně plných listových stránek. Toto prodloužení snižuje efektivitu indexu a plýtvá prostorem na disku.

Zkontrolujte (jako opatření pro úsporu času), zda dávkový aktualizační program volá řádky v pořadí, které definuje index primárního klíče. Toto pořadí způsobí, že se indexové stránky primárního klíče načítají jedna po druhé a každá pouze jednou.

Přítomnost indexů také zpomaluje naplnění tabulek daty pomocí příkazu LOAD nebo obslužného programu **dbload**. Zavedení tabulky, která nemá indexy, je rychlý proces (rychlejší než sekvenční kopie disku na disk), ale aktualizace indexů znamená obrovskou režii.

Zavedení tabulky, která nemá indexy:

1. Vypuštěním zrušte tabulku (pokud existuje).
2. Vytvořte tabulku, aniž byste určili jedinečná omezení.
3. Zaveďte do této tabulky všechny řádky.
4. Změňte tabulku tak, aby používala jedinečná omezení.
5. Vytvořte nejedinečné indexy.

Pokud nemůžete zaručit, že zavedená data vyhovují všem jedinečným indexům, musíte vytvořit jedinečné indexy před zavedením řádků. Ušetříte tak čas, pokud jsou řádky alespoň pro jeden z indexů uvedeny ve správném pořadí. Máte-li možnost volby, zvolte řádek s největším klíčem. Tato strategie minimalizuje počet listových stránek, které musejí být přečteny a zapsány.

Vytvoření a vypuštění indexu v online prostředí

Pomocí příkazů CREATE INDEX ONLINE a DROP INDEX ONLINE můžete vytvořit a vypustit index v online prostředí, kde jsou databáze a její přidružené tabulky neustále dostupné.

Příkaz CREATE INDEX ONLINE umožňuje vytvořit index, aniž byste museli tabulku během vytváření indexů zamknout výlučným zámkem. Příkaz CREATE INDEX ONLINE můžete použít dokonce i tehdy, vyskytují-li se v tabulce čtení i aktualizace. To znamená, že vytváření indexu může začít okamžitě.

Vytváříte-li index online, databázový server tuto operaci protokoluje s příznakem, takže operace ení a obnovení dat mohou index vytvořit znovu.

Při vytváření indexu online můžete omezit velikost paměti, která je přidělena společné oblasti protokolu *preimage* a společné oblasti protokolu *updator* ve sdílené paměti, pomocí

konfiguračního parametru ONLIDX_MAXMEM. To může být užitečné, jestliže plánujete dokončení jiných operací ve sloupci tabulky a současně provádíte příkaz CREATE INDEX ONLINE pro tento sloupec. Další informace o tomto parametru naleznete v části “Omezení přidělování paměti při vytváření indexů online pomocí konfiguračního parametru ONLIDX_MAXMEM” na stránce 7-12.

Příkaz DROP INDEX ONLINE umožňuje vypustit indexy dokonce i tehdy, je-li odstínění transakcí na úrovni neaktualizovaného čtení.

Mezi výhody vytváření indexů pomocí příkazu CREATE INDEX ONLINE patří:

- Potřebujete-li zvýšit výkon dotazů v tabulce pomocí nového indexu, můžete tento index ihned vytvořit bez použití zámku tabulky.
- Databázový server umí vytvořit index i tehdy, jestliže je tabulka právě aktualizována.
- Tabulka je po dobu vytváření indexů dostupná.
- Optimalizátor dotazů může sestavit lepší plány dotazů, protože může aktualizovat statistické údaje v nezamknutých tabulkách.

Mezi výhody vypuštění indexů pomocí příkazu DROP INDEX ONLINE patří:

- Neúčinný index můžete vypustit, aniž byste narušili probíhající dotazy, které tento index používají.
- Optimalizátor dotazů nebude v nových operacích SELECT v tabulkách používat indexy označené příznakem.

Vyvoláte-li příkaz DROP INDEX ONLINE pro tabulku, která je právě aktualizována, operace se neprovede, dokud nebude dokončena aktualizace tabulky. Po vyvolání příkazu DROP INDEX ONLINE již nelze index použít v odkazu, ale souběžné operace mohou index používat, dokud nebudou ukončeny. Databázový server čeká s vypuštěním indexu, dokud všichni uživatelé nedokončí práci s tímto indexem.

Příklad vytvoření indexu v prostředí online:

```
CREATE INDEX idx_1 ON table1(col1) ONLINE
```

Příklad vypuštění indexu v prostředí online:

```
DROP INDEX idx_1 ONLINE
```

Další informace o příkazech CREATE INDEX ONLINE a DROP INDEX ONLINE naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Situace, ve kteých nelze vytvořit ani vypustit indexy online

Pomocí příkazu CREATE INDEX ONLINE nemůžete:

- Vytvořit index, je-li ve stejnou dobu měněna tabulka.
- Vytvořit klastrovaný index.
- Vytvořit index rozhraní VII (Virtual-Index Interface)/R-strom.
- Vytvořit funkční index.
- Vytvořit online příkaz pro index, který je uvnitř transakce.

Pomocí příkazu DROP INDEX ONLINE nemůžete:

- Vypustit index rozhraní VII (Virtual-Index Interface)/R-strom.
- Vypustit klastrovaný index.
- Vypustit online příkaz pro index, který je uvnitř transakce.

Vytvoření připojených indexů v online prostředí

Připojené indexy můžete vytvořit pomocí příkazu CREATE INDEX ONLINE, ale tento příkaz funguje pouze tehdy, je-li odstínění transakcí na úrovni neaktualizovaného čtení. Vytváření indexů zamkne tabulku výlučným zámkem a před vytvořením indexu čeká, až všechny ostatní souběžné procesy, které tuto tabulku prohledávají, ukončí práci s oddíly indexu. Je-li tabulka právě čtena nebo aktualizována, příkaz CREATE INDEX ONLINE čeká na výlučný zámek po dobu nastavení režimu uzamčení.

Omezení přidělování paměti při vytváření indexů online pomocí konfiguračního parametru ONLIDX_MAXMEM

Konfigurační parametr ONLIDX_MAXMEM omezuje velikost paměti, která je přidělena jedné společné oblasti *preimage* a jedné společné oblasti protokolu *updator*. Společné oblasti preimage a updator log, **pimage_<partnum>** a **ulog_<partnum>,** jsou společné oblasti sdílené paměti, které se vytvářejí při provádění příkazu CREATE INDEX ONLINE. Tyto společné oblasti se uvolní, jakmile se dokončí provádění tohoto příkazu.

Výchozí hodnotou konfiguračního parametru ONLIDX_MAXMEM je 5120 kB. Minimální hodnotou, kterou můžete zadat, je 16 kilobajtů; maximální hodnotou je 4294967295 kilobajtů.

Konfigurační parametr ONLIDX_MAXMEM můžete nastavit před spuštěním databázového serveru nebo jej můžete změnit dynamicky pomocí příkazů **onmode -wf** a **onmode -wm**.

Zvýšení výkonu při vytváření indexů

Databázový server používá paralelní zpracování, kdykoliv je to možné, aby snížil dobu odezvy při vytváření indexů. Počet paralelních procesů vychází z počtu fragmentů v indexu a hodnoty proměnné prostředí **PSORT_NPROCS**. Databázový server vytváří index paralelním zpracováním dokonce i tehdy, má-li priorita PDQ hodnotu 0.

Výkon při vytváření indexů můžete často zvýšit pomocí následujících kroků:

1. Nastavte prioritu PDQ na hodnotu větší než 0, abyste získali více paměti než výchozích 128 kB.

   Nastavíte-li prioritu PDQ na hodnotu větší než 0, můžete při vytváření indexů využívat výhod další paměti pro paralelní zpracování.

   Prioritu PDQ můžete nastavit buď pomocí proměnné prostředí **PDQPRIORITY**, nebo příkazem SET PDQPRIORITY v jazyce SQL.
2. Nenastavujte proměnnou prostředí **PSORT_NPROCS**.
3. Doporučuje se, abyste nenastavovali proměnnou prostředí **PSORT_NPROCS**. Máte-li počítač s více procesory, používá databázový server při řazení klíčů indexu dva jednotkové procesy na jedno řazení, není-li proměnná prostředí **PSORT_NPROCS** nastavena. Počet řazení závisí na počtu fragmentů v indexu, na počtu a velikosti klíčů a na hodnotách konfiguračních parametrů paměti dotazů PDQ.
4. Zajistěte přidělení dostatečné paměti a dostatečného dočasného prostoru pro vytvoření celého indexu.
   a. Odhadněte velikost virtuální sdílené paměti, kterou bude databázový server pravděpodobně potřebovat při řazení.

      Další informace naleznete v části “Odhad paměti potřebné pro řazení” na stránce 7-13.
   b. Určete více paměti pomocí konfiguračních parametrů DS_TOTAL_MEMORY a DS_MAX_QUERIES.

c. Není-li k dispozici dostatek paměti, odhadněte velikost dočasného prostoru potřebného pro vytvoření celého indexu.

   Další informace naleznete v části “Odhad dočasného prostoru pro vytváření indexů” na stránce 7-13.

d. Vytvořte velké dočasné prostory dbspaces pomocí obslužného programu **onspaces -t** a zadejte je v konfiguračním parametru utility DBSPACETEMP nebo v proměnné prostředí **DBSPACETEMP**.

   Informace o tom, jak lze optimalizovat dočasné prostory dbspaces naleznete v části “Konfigurace prostorů dbspace pro dočasné tabulky a soubory řazení” na stránce 5-7.

Odhad paměti potřebné pro řazení

Chcete-li vypočítat velikost virtuální sdílené paměti, kterou by databázový server mohl potřebovat při řazení, odhadněte maximální počet řazení, ke kterým by mohlo dojít souběžně, a vynásobte toto číslo průměrným počtem řádků a průměrnou velikostí řádku.

Odhadnete-li například, že by mohlo souběžně dojít ke 30 řazením, průměrná velikost řádku je 200 bajtů a průměrný počet řádků v tabulce je 400, můžete odhadnout velikost sdílené paměti takto:

```
30 řazení * 200 bajtů * 400 řádků = 2400000 bajtů
```

Pomocí konfiguračního parametru DS_NONPDQ_QUERY_MEM můžete konfigurovat paměť řazení pro všechny dotazy kromě těch, které mají prioritu PDQ rovnou 0 (nula). Minimální a současně výchozí hodnotou konfiguračního parametru DS_NONPDQ_QUERY_MEM je 128 kilobajtů. Maximální podporovaná hodnota je 25 procent hodnoty konfiguračního parametru DS_TOTAL_MEMORY. Další informace naleznete v části “Více paměti pro dotazy se hashovacími spojeními, souhrny a dalšími prvky náročnými na paměť” na stránce 13-21.

Je-li priorita PDQ větší než 0, je maximální velikost sdílené paměti, kterou databázový server přiděluje pro řazení, řízena správcem přidělujícím paměť (správce MGM). Správce MGM určuje pomocí nastavení priority PDQ a následujících konfiguračních parametrů, kolik paměti má pro řazení přidělit:

- DS_TOTAL_MEMORY
- DS_MAX_QUERIES
- MAX_PDQPRIORITY

Další informace o přidělování paměti pro paralelní zpracování naleznete v části “Přidělování zdrojů paralelním databázovým dotazům” na stránce 12-7.

Odhad dočasného prostoru pro vytváření indexů

Chcete-li odhadnout velikost dočasného prostoru potřebného pro vytvoření celého indexu, postupujte takto:

1. Sečtěte celkovou šířku indexovaných sloupců nebo hodnot vrácených z uživatelských funkcí. Tato hodnota je dále nazývána *colsize*.
2. Odhadněte velikost charakteristické položky, která má být řazena, pomocí jednoho z následujících vzorců v závislosti na tom, zda je index připojený, nebo ne:
   a. Pro nefragmentovanou tabulku a fragmentovanou tabulku s indexem vytvořeným bez explicitní strategie fragmentace použijte následující vzorec:

      ```
      sizeof_sort_item = keysize + 4
      ```

   b. Pro fragmentovanou tabulku s explicitně fragmentovaným indexem použijte následující vzorec:

```
sizeof_sort_item =
keysize + 8
```

3. Odhadněte počet bajtů potřebných pro řazení pomocí následujícího vzorce:

   ```
   temp_bytes = 2 * (rows *
   sizeof_sort_item)
   ```

   Tento vzorec používá činitel 2, protože všechny položky se při pomocném řazení v dočasném prostoru ukládají dvakrát. Pomocná řazení se provádějí tehdy, neexistuje-li dostatek paměti pro provedení celého řazení v paměti.

   Hodnota položky *rows* je celkový počet řádků v tabulce podle očekávání.

Uložení několika fragmentů indexu do jednoho prostoru dbspace

Do jednoho prostoru dbspace můžete uložit více fragmentů indexu, a tím omezit celkový počet prostorů dbspace, potřebných pro fragmentovanou tabulku. Každý fragment je v prostoru dbspace uložen v samostatném pojmenovaném oddílu. Uložení více fragmentů indexů do jednoho prostoru dbspace zjednodušuje správu prostorů dbspace.

Tuto funkci lze také použít ke zlepšení výkonu dotazů oproti uložení každého fragmentu do jiného prostoru dbspace, pokud je prostor dbspace uložen v rychlejším zařízení.

Další informace naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server* v části týkající se správy oddílů.

Zvýšení výkonu při kontrolách indexů

Obslužný program **oncheck** umožňuje lepší souběžnost pro tabulky, které používají zámek řádků. Jestliže tabulka používá zámek stránek, umístí obslužný program **oncheck** do tabulky sdílený zámek, až bude provádět kontrolu indexů. Sdílené zámky nedovolí ostatním uživatelům provádět v tabulce aktualizace, vkládání ani odstraňování, dokud obslužný program **oncheck** kontroluje nebo tiskne informace indexu.

Jestliže tabulka používá zámek stránek, vrátí databázový server po spuštění obslužného programu **oncheck** bez volby **-x** následující zprávu:

```
WARNING: index check requires a s-lock on stable whose
lock level is page.
```

Podrobnější informace o zámku obslužného programu **oncheck** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Následující souhrn popisuje zámky prováděné během kontrol indexů:

- Při výchozím nastavení neumístí databázový server do tabulky sdílený zámek, jestliže index kontrolujete pomocí voleb **-ci, -cI, -pk, -pK, -pl** nebo **-pL** obslužného programu **oncheck**, pokud tabulka nepoužívá zámek stránek. Kontroluje-li obslužný program **oncheck** indexy tabulky se zámkem stránek, umístí do tabulky sdílený zámek, takže ostatní uživatelé nemohou provádět aktualizace, vkládání ani odstraňování, dokud není tato kontrola dokončena.
- Protože obslužný program **oncheck** neumístí sdílený zámek do tabulek, které během kontroly indexů používají zámky řádků, nemůže být v kontrole indexů tak přesný. Chcete-li mít úplnou jistotu o dokončení kontroly indexů, spusťte obslužný program **oncheck** s volbou **-x**. S volbou **-x** umístí obslužný program **oncheck** do tabulky sdílený zámek a znemožní tak ostatním uživatelům provádět aktualizace, vkládání a odstraňování, dokud nebude tato kontrola dokončena.

Ověřením tabulky systémového katalogu **systables** můžete zjistit aktuální úroveň tabulky (viz následující příkaz jazyka SQL:

```
SELECT locklevel FROM systables
   WHERE tabname = "customer"
```

Pokud ve sloupci **locklevel** nevidíte hodnotu R (u řádku), můžete změnit úroveň zámku (viz následující příkaz jazyka SQL):

```
ALTER TABLE tab1 LOCK MODE (ROW);
```

Při zamykání řádků může dojít k dalším vedlejším účinkům, např. celkové zvýšení používání zamykání. Další informace o úrovních zamykání popisuje Kapitola 8, “Uzamykání”, na stránce 8-1.

Indexy v uživatelských datových typech

Uživatelé mohou definovat vlastní datové typy a funkce, které jsou v těchto typech provozovány. Moduly DataBlade také poskytují rozšířené datové typy a funkce pro databázový server. Indexy lze definovat v následujících druzích uživatelských datových typů:

- Netransparentní datové typy

  Netransparentní datový typ je základním datovým typem, pomocí kterého můžete definovat sloupce stejným způsobem, jako když používáte vestavěné datové typy. Netransparentní datový typ ukládá jedinou hodnotu a databázový server jej nemůže rozdělit na komponenty. Další informace o vytváření netransparentních datových typů naleznete v příručkách *IBM Informix Guide to SQL: Syntax* a *IBM Informix User-Defined Routines and Data Types Developer's Guide* v částech týkajících se příkazu CREATE OPAQUE TYPE. Další informace o datových typech a funkcích, které každý modul DataBlade poskytuje, naleznete v uživatelské příručce příslušného modulu DataBlade.

- Datový typ distinct

  Datový typ distinct má stejné zastoupení jako již existující netransparentní nebo vestavěný datový typ, ale od těchto typů se odlišuje. Další informace o datových typech distinct naleznete v příručkách *IBM Informix Guide to SQL: Reference* a *IBM Informix Guide to SQL: Syntax* v části týkající se příkazu CREATE DISTINCT TYPE.

Další informace o datových typech naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Definování indexů pro uživatelské datové typy

Možná budete chtít stejně jako v případě vestavěných datových typů při definování indexů pro nové datové typy zvýšit dobu odezvy dotazu. Doba odezvy dotazu se může zvýšit, používá-li Dynamic Server index pro:

- Sloupce použité ke spojení dvou tabulek
- Sloupce, které jsou filtry dotazu
- Sloupce v klauzuli ORDER BY nebo GROUP BY
- Výsledky funkcí, které jsou filtry dotazu

Další informace o tom, kdy lze výkon dotazu zvýšit pomocí indexu nebo vestavěného datového typu, naleznete v části “Zvýšení výkonu pomocí indexů” na stránce 13-14.

Dynamic Servera moduly DataBlade poskytují celou škálu rozdílných typů indexů (také označované jako *sekundární přístupové metody*). Sekundární přístupová metoda je sadou funkcí databázového serveru, která vytváří strukturu indexu, přistupuje k ní a manipuluje s ní. Tyto funkce shrnují indexové operace, např. postupy prohledávání, vkládání, odstraňování či aktualizování uzlů v indexu.

Chcete-li vytvořit index nebo uživatelský datový typ, můžete použít kteroukoliv z následujících sekundárních přístupových metod:

- Obecný index B-stromu

  Index B-stromu je vhodný u dotazu, který vyhledává úsek datových hodnot. Další informace naleznete v části “Sekundární přístupová metoda B-stromu” na stránce 7-16.
- Index R-stromu

  Index R-stromu je vhodný pro vyhledávání ve vícerozměrných datech. Další informace naleznete v příručce *IBM Informix R-Tree Index User's Guide*.
- Sekundární přístupové metody, které zajišťuje modul DataBlade pro nový datový typ

  Modul DataBlade, který podporuje určitý datový typ, pro něj může také poskytnout nový index. Další informace naleznete v části “Použití indexu modulu DataBlade” na stránce 7-20.

V jednom nebo více sloupcích můžete vytvořit funkční index výsledných hodnot uživatelské funkce. Další informace naleznete v části “Použití funkčního indexu” na stránce 7-18.

Po zvolení požadovaného typu indexu budete možná pro sekundární přístupovou metodu potřebovat rozšířit třídu operátorů. Další informace o rozšíření tříd operátorů naleznete v příručce*IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Sekundární přístupová metoda B-stromu

Dynamic Server poskytuje *obecný index B-stromu* pro sloupce v databázových tabulkách. V obvyklých relačních systémech ovládá přístupová metoda B-stromu pouze vestavěné datové typy, a proto může porovnávat pouze dva klíče vestavěných datových typů. Obecný index B-stromu je rozšířenou verzí B-stromu, pomocí kterého Dynamic Server podporuje uživatelské datové typy.

Tip: Další informace o struktuře indexu B-stromu a o odhadnutí jeho velikosti naleznete v části “Odhad indexových stránek” na stránce 7-1**.**

Dynamic Server používá obecný B-strom jako vestavěnou sekundární přístupovou metodu. Tato vestavěná sekundární přístupová metoda je registrovaná v tabulce systémového katalogu **sysams** pod názvem **btree**. Pokud použijete pro vytvoření indexu příkaz CREATE INDEX (bez použití klauzule USING), vytvoří databázový server obecný index B-stromu. Další informace naleznete v příručce *IBM Informix Guide to SQL: Syntax* v části týkající se příkazu CREATE INDEX.

Tip: *Dynamic Server* také definuje další sekundární přístupovou metodu - index R-stromu. Další informace o použití indexu R-stromu naleznete v příručce *IBM Informix R-Tree Index User's Guide*.

Použití indexu B-stromu: Index B-stromu je vhodný u dotazu, který vyhledává úsek datových hodnot. Pokud data, která mají být indexována, obsahují logickou posloupnost, u které lze použít koncepce *méně než*, *více než* a *rovno*, je obecný index B-stromu vhodným způsobem, jak data indexovat. Obecný index B-stromu ve výchozím nastavení u všech vestavěných datových typů podporuje relační operátory (<,<=,=,>=,>) a řadí data v lexikografické posloupnosti.

Optimalizátor zváží, zda k provedení dotazu použít index B-stromu, pokud definujete obecný index B-stromu:

- Pro sloupce použité ke spojení dvou tabulek
- Pro sloupce, které jsou filtry dotazu
- Pro sloupce v klauzuli ORDER BY nebo GROUP BY
- Pro výsledky funkcí, které jsou filtry dotazu

Rozšíření obecného indexu B-stromu: Ve výchozím nastavení může obecný B-strom indexovat data, která jsou jedním z vestavěných datových typů. Tento B-strom řadí data v lexikografické posloupnosti. Obecný B-strom však můžete rozšířit tak, aby podporoval sloupce a funkce u následujících datových typů:

- *Uživatelské datové typy* (netransparentní datové typy a datové typy distinct), u kterých chcete, aby je index B-stromu podporoval.

  V tomto případě je třeba rozšířit výchozí třídu operátorů obecného indexu B-stromu.
- *Vestavěné datové typy,* které chcete uspořádat v jiné sekvenci než lexikografické sekvenci používané obecným indexem B-stromu.

  V tomto případě je třeba definovat jinou třídu operátorů než výchozí obecný index B-stromu.

Třída operátorů je sadou funkcí (operátorů), které jsou přidružené k netradičnímu indexu B-stromu. Další informace o třídách operátorů naleznete v části “Zvolení tříd operátorů pro indexy” na stránce 7-20.

Určování dostupných přístupových metod

Dynamic Server používá vestavěný B-strom jako sekundární přístupovou metodu. Prostředí, které používáte, možná nainstalovalo moduly DataBlade, které implementují další sekundární přístupové metody. Pokud existují další sekundární přístupové metody, jsou definované v tabulce systémového katalogu **sysams**.

Chcete-li určit sekundární přístupové metody dostupné pro vaši databázi, dotazujte se tabulky systémového katalogu **sysams** pomocí následujícího příkazu SELECT:

```
SELECT am_id, am_owner, am_name, am_type FROM sysams
   WHERE am_type = 'S';
```

Hodnota S ve sloupci **am_type** identifikuje přístupovou metodu jako sekundární. Tento dotaz vrátí následující informace:

- Sloupce **am_id** a **am_name** identifikují sekundární přístupovou metodu.
- Sloupec **am_owner** identifikuje vlastníka přístupové metody.

V databázi kompatibilní se standardem ANSI musí být název přístupové metody v rámci oboru názvů uživatele jedinečný. Název přístupové metody vždy začíná uživatelem ve formátu **am_owner.am_name**.

Ve výchozím nastavení používá Dynamic Server v tabulce systémového katalogu **sysams** definice pro dvě sekundární přístupové metody - **btree** a **rtree**.

Přístupová metoda	Sloupec am_id	Sloupec am_name	Sloupec am_owner
Obecný B-strom	1	btree	’informix’
R-strom	2	rtree	’informix’

Důležité: Tabulka systémového katalogu **sysams** neobsahuje řádek pro vestavěnou primární přístupovou metodu. Tato primární přístupová metoda je pro Dynamic Server interní a nevyžaduje definici v tabulce **sysams**. Vestavěná primární přístupová metoda je však vždy k dispozici.

Pokud v tabulce systémového katalogu **sysams** naleznete další řádky (řádky s hodnotami **am_id**, které jsou vyšší než 2), databázový server podporuje další uživatelské přístupové metody. Kontrolou hodnoty ve sloupci **am_type** zjistíte, zda je uživatelská přístupová metoda primární nebo sekundární.

Další informace o sloupcích v tabulce systémového katalogu **sysams** naleznete v příručce *IBM Informix Guide to SQL: Reference*. Další informace o určení tříd operátorů dostupných ve vaší databázi naleznete v části “Určení dostupných tříd operátorů” na stránce 7-22.

Použití uživatelské sekundární přístupové metody

Vestavěná sekundární přístupová metoda je index B-stromu. Pokud se koncepty *méně než*, *více než* a *rovno* nevztahují na indexovaná data, pravděpodobně budete chtít zvážit použití *uživatelské sekundární přístupové metody*, která funguje s Dynamic Server. K přístupu k dalším indexovacím strukturám (např. k indexu R-stromu) je možné použít uživatelskou sekundární přístupovou metodu.

Pokud databáze podporuje uživatelskou sekundární přístupovou metodu, můžete specifikovat, že databázový server používá tuto přístupovou metodu při přístupu ke konkrétnímu indexu. Další informace o určení sekundárních přístupových metod definovaných databází naleznete v části “Určování dostupných přístupových metod” na stránce 7-17.

Pomocí klauzule USING příkazu CREATE INDEX zvolíte uživatelskou sekundární přístupovou metodu. Klauzule USING určuje název sekundární přístupové metody, který se má použít při vytváření indexu. Tento název musí být uveden ve sloupci **am_name** tabulky systémového katalogu **sysams** a musí se jednat o sekundární přístupovou metodu (sloupec **am_type** tabulky **sysams** je 'S').

Sekundární přístupová metoda, kterou určíte pomocí klauzule USING příkazu CREATE INDEX musí být již definovaná v systémovém katalogu **sysams**. Pokud ještě nebyla sekundární přístupová metoda definována, příkaz CREATE INDEX se nezdaří.

Pokud z příkazu CREATE INDEX vypustíte klauzuli USING, jako sekundární přístupovou metodu použije databázový server indexy B-stromu. Další informace naleznete v příručce *IBM Informix Guide to SQL: Syntax* v části týkající se příkazu CREATE INDEX.

Index R-stromu: Dynamic Server podporuje *index R-stromu* u sloupců, které obsahují prostorová data (např. mapy a diagramy). Index R-stromu používá stromovou strukturu, jejíž uzly ukládají ukazatele do uzlů nižší úrovně. V listech R-stromu se nacházejí kolekce datových stránek, které ukládají *n*-rozměrné tvary. Další informace o struktuře indexu R-stromu a o postupu odhadnutí jeho velikosti naleznete v příručce *IBM Informix R-Tree Index User's Guide*.

Použití funkčního indexu

Dynamic Server podporuje indexy následujících databázových objektů:

- Sloupcový index

  Sloupcový index můžete vytvořit v aktuálních hodnotách jednoho nebo více sloupců.
- Funkční index

  Funkční index můžete vytvořit ve funkční hodnotě jednoho nebo více sloupců.

Důležité: Nelze vytvořit funkční index ve funkční hodnotě sloupce, který obsahuje datový typ kolekce.

Rozhodujete-li se, zda použít sloupcový nebo funkční index, zjistěte, zda je sloupcový index vhodný pro data, která si přejete indexovat. Index ve sloupci některých datových typů nemusí být pro běžné dotazy potřebný. Následující dotaz se například ptá, kolik obrazů je tmavých:

```
SELECT COUNT(*) FROM photos WHERE
darkness(picture) > 0.5
```

Samotný index dat **obrázku** nezvyšuje výkon dotazu. Koncepce *menší než*, *více než* a *rovno* nemají význam obzvláště při jejich použití na obrázkový datový typ. Namísto toho může

zvýšit výkon funkční index používající funkci **darkness()**. Můžete mít také takovou uživatelskou funkci, k jejímuž vykonávání dochází dostatečně často, a tak při vytvoření indexu na jejích hodnotách dojde ke zvýšení výkonu.

Co je to funkční index: Při vytváření funkčního indexu databázový server počítá hodnoty uživatelské funkce a ukládá je jako klíčové hodnoty v indexu. Pokud změna v datech tabulky způsobí změnu v jedné z hodnot indexového klíče, databázový server automaticky aktualizuje funkční index.

Funkční index je možné použít u funkcí, které vrátí hodnoty uživatelských datových typů (netransparentních a typu distinct) i vestavěných datových typů. V případě, že funkce vrátí datový typ jednoduchého velkého objektu (TEXT nebo BYTE), nelze funkční index definovat.

Kdy se funkční index používá: Optimalizátor zváží, zda použít funkční index k přístupu k výsledkům funkcí, které jsou v jedné z následujících klauzulí dotazu:

- Klauzule SELECT
- Filtry v klauzuli WHERE

Funkční index může být index B-stromu, index R-stromu nebo typ uživatelského indexu, který poskytuje modul DataBlade. Další informace o typech indexů naleznete v části “Definování indexů pro uživatelské datové typy” na stránce 7-15. Další informace o požadavcích na prostor pro funkční indexy naleznete v části “Odhadování indexových stránek” na stránce 3-13.

Jak vytvořit funkční index: Funkce může být vestavěná nebo uživatelská. Uživatelská funkce může být externí funkcí nebo funkcí SPL.

Postup vytvoření funkčního indexu pro uživatelskou funkci:

1. Pokud se jedná o externí funkci, napište pro uživatelskou funkci kód.
2. Pomocí příkazu CREATE FUNCTION registrujte uživatelskou funkci v databázi.
3. Pomocí příkazu CREATE INDEX vytvořte funkční index.

Postup vytvoření funkčního indexu pro funkci darkness():

1. Napište kód pro uživatelskou funkci **darkness()**, která operuje s datovým typem a navrátí dekadickou hodnotu.
2. Pomocí příkazu CREATE FUNCTION registrujte uživatelskou funkci v databázi:

   ```
   CREATE FUNCTION darkness(im image)
   RETURNS decimal
   EXTERNAL NAME '/lib/image.so'
   LANGUAGE C NOT VARIANT
   ```

   V tomto příkladu můžete pro funkční index použít výchozí třídu operátorů, protože vrácená hodnota funkce **darkness()** je vestavěným datovým typem, DECIMAL.
3. Pomocí příkazu CREATE INDEX vytvořte funkční index.

   ```
   CREATE TABLE photos
   (
       name char(20),
       picture image
   ...
   );
   CREATE INDEX dark_ix ON photos (darkness(picture));
   ```

   V tomto příkladu předpokládejme, že byl uživatelský datový typ **image** již definován v databázi.

Optimalizátor nyní může zvážit funkční index, pokud určíte funkci **darkness()** jako filtr v dotazu:

```
SELECT count(*) FROM photos WHERE
darkness(picture) > 0.5
```

Pomocí uživatelských funkcí můžete také vytvořit složený index. Další informace naleznete v části “Použití složených indexů” na stránce 13-14.

Upozornění: Nevytvářejte funkční index pomocí funkcí DECRYPT_BINARY() ani DECRYPT_CHAR(). Tyto funkce v databázi ukládají prostá textová data, čímž ruší účel šifrování. Další informace o šifrování naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Použití indexu modulu DataBlade

V systému Dynamic Server mají uživatelé přístup k novým datovým typům, které poskytují moduly DataBlade. Modul DataBlade může také poskytovat nový index pro nový datový typ. Například pomocí modulu DataBlade Excalibur Text Search může index prohledávat textová data. Další informace naleznete v příručce *Excalibur Text Search DataBlade Module User's Guide*.

Další informace o typech dat a funkcí, které každý modul DataBlade poskytuje, naleznete v uživatelské příručce modulu DataBlade. Další informace o určování typů indexů dostupných ve vaší databázi naleznete v části “Určování dostupných přístupových metod” na stránce 7-17.

Zvolení tříd operátorů pro indexy

Ve většiny situací používejte výchozí operátory, které jsou definované pro sekundární přístupovou metodu. V případě, že budete chtít uspořádat data v jiné posloupnosti nebo pomocí indexů podpořit uživatelské datové typy, bude však nutné rozšířit třídu operátorů. Další informace o rozšíření třídy operátorů naleznete v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Třídy operátorů

Třída operátorů je sada názvů funkcí, která je přidružená k sekundární přístupové metodě. Tyto funkce umožňují sekundární přístupové metodě ukládat a vyhledávat hodnoty konkrétního datového typu. Optimalizátor dotazů databázového serveru pomocí třídy operátorů určuje, zda může index zpracovat dotaz s nejnižšími náklady. Třída operátorů oznamuje optimalizátoru dotazů dvě věci:

- Které funkce zobrazené v příkazu SQL lze vyhodnotit pomocí daného indexu

  Tyto funkce se nazývají *strategické funkce* třídy operátorů.
- Pomocí kterých funkcí index vyhodnocuje strategické funkce

  Tyto funkce se nazývají *podpůrné funkce* třídy operátorů.

Pomocí informací od třídy operátorů může optimalizátor dotazů určit, zda je daný index pro dotaz použitelný. Optimalizátor dotazů může zvážit, zda pro daný dotaz index použít, jsou-li splněny následující podmínky:

- Index se v dotazu nachází na konkrétním sloupci nebo sloupcích.
- U existujícího indexu odpovídá operace ve sloupci nebo sloupcích dotazu jedné ze strategických funkcí ve třídě operátorů, které jsou přidružené k indexu.

Optimalizátor dotazů zkontroluje dostupné indexy pro tabulku nebo tabulky a sloučí indexové klíče se sloupcem definovaným ve filtru dotazů. Pokud sloupec ve filtru odpovídá indexovému klíči a funkce ve filtru je jednou ze strategických funkcí třídy operátorů, zahrne

optimalizátor index po zjištění, který plán dotazů má nejnižší náklady na provedení. Tímto způsobem může optimalizátor určit, který index může zpracovat dotaz za co nejnižších nákladů.

Dynamic Server ukládá informace o třídách operátorů v tabulce systémového katalogu **sysopclasses**.

Strategické a podpůrné funkce: Dynamic Server prostřednictvím *strategických funkcí* sekundární přístupové metody pomáhá optimalizátoru dotazů určit, zda je specifický index použitelný na specifickou operaci datového typu. Pokud existuje index a operátor ve filtru odpovídá jedné ze strategických funkcí ve třídě operátorů, optimalizátor zváží, zda pro dotaz tento index použít.

Dynamic Server pomocí *podpůrných funkcí* sekundární přístupové metody vytváří a přistupuje k indexům. Tyto funkce nejsou volány přímo koncovými uživateli. Pokud operátor ve filtru dotazu sloučí jednu z následujících strategických funkcí, použije sekundární přístupová metoda podpůrné funkce, pomocí kterých překročí index a získá výsledky. Identifikace aktuálních podpůrných funkcí je ponechána sekundární přístupové metodě.

Výchozí třídy operátorů: Každá sekundární přístupová metoda obsahuje přidruženou *výchozí třídu operátorů*. Ve výchozím nastavení příkaz CREATE INDEX přidružuje výchozí třídu operátorů k indexu. Například následující příkaz CREATE INDEX vytvoří index B-stromu ve sloupci **postalcode** a automaticky k tomuto sloupci přidruží výchozí třídu operátorů B-stromu:

```
CREATE INDEX postal_ix ON customer(postalcode)
```

Další informace o stanovení nové třídy operátorů pro index naleznete v části “Použití třídy operátorů” na stránce 7-23.

Třída operátorů vestavěného B-stromu

Vestavěná sekundární přístupová metoda, obecný B-strom, obsahuje výchozí třídu operátorů nazývanou **btree_ops**. Tato třída je definovaná v tabulce systémového katalogu **sysopclasses**. Ve výchozím nastavení příkaz CREATE INDEX přidružuje třídu operátorů **btree_ops** k indexu B-stromu. Například následující příkaz CREATE INDEX vytváří obecný index B-stromu ve sloupci **order_date** tabulky **orders** a přidružuje k tomuto indexu výchozí třídu operátorů pro sekundární přístupovou metodu B-stromu:

```
CREATE INDEX orddate_ix ON orders (order_date)
```

Dynamic Server pomocí třídy operátorů **btree_ops** specifikuje:

- Strategické funkce sdělující optimalizátoru dotazů, které filtry v dotazu mohou používat index B-stromu.
- Podpůrnou funkci, pomocí které bude vytvořen a vyhledán index B-stromu.

Strategické funkce B-stromu: Třída operátorů **btree_ops** definuje následující názvy strategických funkcí pro přístupovou metodu **btree**:

- **lessthan** (<)
- **lessthanorequal** (<=)
- **equal** (=)
- **greaterthanorequal** (>=)
- **greaterthan** (>)

Veškeré tyto strategické funkce jsou *funkcemi operátorů*. To znamená, že je každá funkce přidružená k symbolu operátoru - v tomto případě k symbolu relačního operátoru. Další informace o funkcích relačního operátoru naleznete v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Optimalizátor dotazů při kontrole dotazu obsahujícího sloupec také kontroluje, zda je v tomto sloupci definovaný index B-stromu. Pokud takový index existuje *a* pokud dotaz obsahuje jeden z relačních operátorů, které podporuje třída operátorů **btree_ops**, optimalizátor může pro provedení dotazu zvolit index B-stromu.

Podpůrná funkce B-stromu: Třída operátorů **btree_ops** obsahuje podpůrnou porovnávací funkci **compare()**. Funkce **compare()** je uživatelskou funkcí, která vrátí celočíselnou hodnotu, pomocí které zjistí, zda je její první argument roven, menší než nebo vyšší než druhý argument následujícím způsobem:

- Hodnota 0, je-li první argument *roven* druhému argumentu
- Hodnota menší než 0, je-li první argument *menší než* druhý argument
- Hodnota větší než 0, je-li první argument *vyšší než* druhý argument

Sekundární přístupová metoda B-stromu pomocí funkce **compare()** přeskočí uzly obecného indexu B-stromu. Aby bylo možné v obecném indexu B-stromu vyhledávat datové hodnoty, sekundární přístupová metoda pomocí funkce **compare()** porovnává hodnotu klíče v dotazu s hodnotou klíče v uzlu indexu. Výsledek tohoto porovnání určí, zda sekundární přístupová metoda potřebuje vyhledávat další index na nižší úrovni nebo zda se klíč nachází v aktuálním uzlu.

Přístupová metoda obecného B-stromu také pomocí funkce **compare()** provádí pro indexy obecného B-stromu následující úlohy:

- Seřazuje klíče ještě před vytvořením indexu.
- Určuje lineární uspořádání klíčů v indexu obecného B-stromu.
- Vyhodnocuje relační operátory.
- Vyhledává datové hodnoty v indexu.

Databázový server pomocí funkce **compare()** vyhodnocuje výsledky porovnávání v příkazu SELECT. Chcete-li podpořit tyto výsledky porovnávání u netransparentních datových typů, je třeba zadat funkci **compare()** . Další informace naleznete v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Databázový server také v případě použití indexu B-stromu pomocí funkce **compare()** zpracovává klauzuli ORDER BY v příkazu SELECT. Pokud však index nepoužívá třídu operátorů btree-ops, optimalizátor neprovede operaci ORDER BY pomocí indexu.

Určení dostupných tříd operátorů

Databázový server poskytuje výchozí třídu operátorů pro vestavěnou sekundární přístupovou metodu, index obecného B-stromu. Prostředí, které používáte, navíc možná nainstalovalo moduly DataBlade, které implementují další třídy operátorů. Všechny třídy operátorů jsou definovány v tabulce systémového katalogu **sysopclasses**.

Chcete-li určit třídy operátorů dostupné pro vaši databázi, dotazujte se tabulky systémového katalogu **sysopclasses** pomocí následujícího příkazu SELECT:

```
SELECT opclassid, opclassname, amid, am_name
FROM sysopclasses, sysams
WHERE sysopclasses.amid = sysams.am_id
```

Tento dotaz vrátí následující informace:

- Sloupce **opclassid** a **opclassname** identifikují třídu operátorů.
- Sloupce **am_id** a **am_name** identifikují přidružené sekundární přístupové metody.

Ve výchozím nastavení používá databázový server v tabulce systémového katalogu **sysopclasses** následující definice pro dvě třídy operátorů, **btree_ops** a **rtree_ops**.

Přístupová metoda	Sloupec opclassid	Sloupec opclassname	Sloupec amid	Sloupec am_name
Obecný B-strom	1	btree_ops	1	btree
R-strom	2	rtree_ops	2	rtree

Pokud v tabulce systémového katalogu **sysopclasses** naleznete další řádky (řádky s hodnotami **opclassid**, které jsou vyšší než 2), podporuje databáze uživatelské třídy operátorů. Kontrolou hodnoty ve sloupci **amid** určíte sekundární přístupové metody, ke kterým třída operátorů náleží.

Sloupec **am_defopclass** v tabulce systémového katalogu **sysams** ukládá identifikátor třídy operátorů pro výchozí třídu operátorů sekundární přístupové metody. Chcete-li určit výchozí třídu operátorů u dané sekundární přístupové metody, můžete spustit následující dotaz:

```
SELECT am_id, am_name, am_defopclass, opclass_name
FROM sysams, sysopclasses
WHERE sysams.am_defopclass = sysopclasses.opclassid
```

Databázový server ve výchozím nastavení poskytuje následující výchozí třídy operátorů.

Přístupová metoda	Sloupec am_id	Sloupec am_name	Sloupec am_defopclass	Sloupec opclass_name
Obecný B-strom	1	btree	1	btree_ops
R-strom	2	rtree	2	rtree_ops

Další informace o sloupcích tabulek systémového katalogu **sysopclasses** a **sysams** naleznete v příručce *IBM Informix Guide to SQL: Reference*. Další informace o určení přístupových metod dostupných v databázi naleznete v části “Určování dostupných přístupových metod” na stránce 7-17.

Použití třídy operátorů

Příkaz CREATE INDEX určuje, která třída operátorů se má použít pro každou komponentu indexu. Pokud neurčíte třídu operátorů, příkaz CREATE INDEX pro vytvořenou sekundární přístupovou metodu použije výchozí třídu operátorů. Pro komponenty indexu můžete použít uživatelskou třídu operátorů. Chcete-li určit uživatelskou třídu operátorů pro konkrétní komponentu indexu, můžete postupovat takto:

- Použijte uživatelskou třídu operátorů, kterou databáze také definuje.
- Použijte třídu operátorů R-stromu, pokud databáze definovala sekundární přístupovou metodu R-stromu. Další informace o R-stromech naleznete v příručce *IBM Informix R-Tree Index User's Guide*.

Pokud databáze u sekundární přístupové metody podporuje více tříd operátorů, můžete určit, které třídy operátorů databázový server pro konkrétní index použije. Další informace o určení tříd operátorů definovaných databází naleznete v části “Určení dostupných tříd operátorů” na stránce 7-22.

Každá část složeného indexu může určit jinou třídu operátorů. Třídy operátorů vybíráte při tvorbě indexu. V příkazu CREATE INDEX určete název třídy operátorů, který se má po

každém názvu sloupce nebo funkce ve specifikaci indexového klíče použít. Každý název musí být uveden ve sloupci **opclassname** tabulky systémového katalogu **sysopclasses** a musí být přidružen k sekundární přístupové metodě, kterou index používá.

Pokud například databáze pomocí sekundární přístupové metody **abs_btree_ops** definuje nové pořadí řazení, následující příkaz CREATE INDEX určí, že tabulka **table1** přidružuje třídu operátorů **abs_btree_ops** k indexu B-stromu **col1_ix**:

```
CREATE INDEX col1_ix ON table1(col1 abs_btree_ops)
```

Třída operátorů určená v příkazu CREATE INDEX musí být již definovaná v systémovém katalogu **sysopclasses** pomocí příkazu CREATE OPCLASS statement. Pokud nebyla třída operátorů dosud definována, spuštění příkazu CREATE INDEX se nezdaří. Další informace o vytvoření tříd operátorů naleznete v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Kapitola 8. Uzamykání

Obsah kapitoly

Tato kapitola se zabývá použitím zámků, vlivem zámků na souběžnost a výkon a sledováním a správou zámků.

Zámek

Zámek je softwarový mechanismus, který zabraňuje ostatním uživatelům v používání zdroje. Zámek můžete umístit do následujících položek:

- Jednotlivý řádek nebo klíč
- Stránka dat nebo klíčů indexu
- Tabulka
- Databáze

Pro inteligentní velké objekty jsou dostupné další typy zámků. Více informací naleznete v části “Zámky inteligentních velkých objektů” na stránce 8-14.

Maximální počet řádků nebo stránek zamčených v jedné transakci je řízen celkovým počtem nakonfigurovaných zámků. Počet tabulek v nichž jsou tyto řádky nebo stránky zamčeny není explicitně určen.

Granularita zámku

Úroveň a typ informace, jíž zámek chrání, se nazývá *granularita zamykání*. Granularita zamykání ovlivňuje výkon. Pokud uživatel nemůže přistupovat k řádku nebo klíči, může počkat, dokud jiný uživatel tento řádek nebo klíč neodemkne. Pokud uživatel zamkne celou stránku, je pravděpodobné, že na řádek v této stránce bude čekat více uživatelů. Vlastnost umožňující více uživatelům přistupovat k sadě řádků se nazývá *souběžnost*. Cílem administrátora databáze je zvýšit souběžnost, a tím i celkový výkon, bez snížení výkonu pro jednotlivé uživatele.

Zámky řádků a klíčů

Zamykání řádku a klíče není výchozím chováním. Výchozí režim uzamčení je režim zamykání stránek, jak vysvětluje "Zámky stránek" na stránce 8-3. Musíte vytvořit tabulku s nastaveným uzamykáním řádků, jak ukazuje následující příklad:

```
CREATE TABLE customer(customer_num serial, lname char(20)...)
   LOCK MODE ROW;
```

Režim uzamčení můžete změnit příkazem ALTER TABLE.

Je-li režim uzamčení nastaven na možnost ROW a vložíte nebo aktualizujete řádek, databázový server vytvoří zámek řádku. V některých případech umístíte zámek řádku jednoduše přečtením řádku příkazem SELECT.

Je-li režim uzamčení nastaven na možnost ROW a vložíte, aktualizujete nebo odstraníte klíč (provede se automaticky, pokud vložíte, aktualizujete nebo odstraníte řádek), databázový server vytvoří v indexu zámek klíče.

Výhody a nevýhody zamykání řádku a klíče

Zámky řádku a klíče obecně poskytují nejlepší celkový výkon pokud aktualizujete relativně malý počet řádků, protože zvyšují souběžnost, ovšem databázový server může být při získávání zámků zahlcen. U operací měnících velký počet řádků může být získání zámku pro každý řádek neefektivní. V takovém případě zvažte použití zámku stránky.

Zámky hodnoty klíče

Odstraní-li uživatel během transakce řádek, nemůže být tento řádek zamčen, protože neexistuje. Databázový server ovšem musí do konce transakce existenci řádku zaznamenávat.

Databázový server používá koncepci zvanou *zamykání hodnoty klíče* k zamčení odstraněných řádků. Když databázový server odstraní řádek, hodnoty klíče v indexu tabulky nejsou odstraněny okamžitě. Místo toho je každá hodnota klíče označena jako odstraněná a je do ní umístěn zámek.

Ostatní uživatelé mohou narazit na hodnotu klíče označenou jako odstraněná. Databázový server musí určit jestli zámek existuje, a pokud ano, transakce odstranění nebyla potvrzena a databázový server odešle chybu uzamčení zpět aplikaci (nebo počká než bude zámek uvolněn, pokud uživatel provedl operaci SET LOCK MODE TO WAIT).

Jedním z nejdůležitějších použití zamykání hodnoty klíče je zajištění jedinečnosti klíče do konce transakce, která ho odstranila. Bez tohoto ochranného mechanismu by mohl uživatel A odstranit jedinečný klíč a uživatel B vložit řádek se stejným klíčem před potvrzením

transakce uživatele A. V takovém případě by uživatel A nemohl transakci odvolat. Zamykání hodnoty klíče zabraňuje uživateli B ve vložení řádku do konce transakce uživatele A.

Zámky stránek

Zamykání stránek je výchozím chováním při vytváření tabulek bez klauzule LOCK MODE. Při použití zamykání stránek zamkne databázový server namísto řádku celou stránku obsahující tento řádek. Aktualizuje-li několik řádků na jedné stránce, použije databázový server pro uzamčení celé stránky jen jeden zámek.

Pokud vkládáte nebo aktualizujete řádek, vytvoří databázový server zámek stránky na stránce s daty. V některých případech vytvoří databázový server zámek stránky při přečtení řádku příkazem SELECT.

Pokud vložíte, aktualizujete nebo odstraníte klíč (provede se automaticky, když vložíte, aktualizujete nebo odstraníte řádek), databázový server uzamkne v indexu stránku obsahující klíč.

Důležité: Zámek stránky umístěný do indexové stránky může snížit souběžnost podstatně více než zámek stránky umístěný do stránky s daty. Stránky indexu jsou hustě zaplněny a obsahují velký počet klíčů. Zamknutím indexové stránky můžete teoreticky ostatním uživatelům až do uvolnění zámku učinit nedostupný velký počet klíčů. Tabulky používající zámky stránek nemohou podporovat vlastnost souběžnosti USELASTCOMMITTED, která je popsána v části “Úroveň izolace potvrzené čtení” na stránce 8-6.

Zámky stránek jsou užitečné v tabulkách, kde běžný uživatel mění velký počet řádků najednou. Například tabulky, které obsahují objednávky a je do nich často vkládáno a z nich dotazováno po jedné položce, nejsou vhodné pro použití zamykání stránek. Naproti tomu tabulky, které obsahují staré objednávky a jsou aktualizovány v nočních hodinách objednávkami učiněnými během dne, jsou pro použití zámků stránek vhodným kandidátem. V tomto případě je typ použité úrovně izolace důležitý. Další informace naleznete v části “Úroveň izolace” na stránce 8-5.

Zámky tabulky

V prostředí datových skladů může být pro dotazování vhodnější použít zámek s větší granularitou. Přistupuje-li například dotaz k většině řádků v tabulce, zvýší se jeho efektivita použitím menšího počtu zámků tabulky namísto mnoha zámků stránek nebo řádků.

Databázový server může umístit dva typy zámků tabulky:

- Sdílený zámek

  Žádní další uživatelé nemohou do tabulky zapisovat.
- Výlučný zámek

  Žádní další uživatelé nemohou z tabulky číst ani do ní zapisovat.

Dalším důležitým rozdílem těchto dvou typů zámků tabulek je skutečný počet umístěných zámků:

- Ve sdíleném režimu umístí databázový server do tabulky jeden sdílený zámek, který informuje ostatní uživatele o nemožnosti provádět aktualizace. Navíc za každý aktualizovaný, odstraněný nebo vložený řádek přidá databázový server další zámek.
- Ve výlučném režimu umístí databázový server do tabulky jen jeden výlučný zámek bez ohledu na to, kolik řádků je aktualizáno. Jetliže aktualizujete většinu řádek, umístěte do tabulky výlučný zámek.

Důležité: Tabulkový zámek může výrazně snížit souběžnost aktualizací tabulky. V jednom okamžiku může k tabulce přistupovat jen jedna aktualizační transakce, která zablokuje všechny ostatní transakce. Vyjímku tvoří transakce jen pro čtení, které mohou přistupovat k tabulce simultánně. Toto chování je užitečné v prostředí datových skladů, kde jsou zavedená data dotazována více uživateli.

Úroveň zamčení tabulky je možné přepnout mezi tabulkovým zámkem a jinými úrovněmi zámků. Schopnost přepnout úroveň zámku je užitečná, pokud používáte tabulku po určité období v režimu datového skladu.

Transakce sdělí databázovému serveru příkazem LOCK TABLE, jestli má použít zámek na úrovni tabulky. V následujícím příkladě je do tabulky umístěn výlučný zámek:

```
LOCK TABLE tab1 IN EXCLUSIVE MODE;
```

V následujícím příkladě je do tabulky umístěn sdílený zámek:

```
LOCK TABLE tab1 IN SHARE MODE;
```

V některých případech umístí databázový server vlastní tabulkový zámek. Je-li například úroveň izolace nastavena na opakovatelné čtení a databázový server musí přečíst velkou část tabulky, umístí automaticky tabulkový zámek namísto zámků řádků nebo stránek. Tabulkový zámek je také umístěn do tabulky při vytváření nebo vypuštění indexu.

Databázové zámky

Při otevření databáze do ní můžete umístit zámek použitím příkazu DATABASE, čímž uzamknete celou databázi. Databázový zámek zabraňuje všem ostatním uživatelům číst a aktualizovat data.

Následující příkaz otevře a zamkne databázi sales:

```
DATABASE sales EXCLUSIVE
```

Konfigurace režimu uzamčení

Výchozí režim uzamčení při vytváření tabulky je page. Pokud ke stejné tabulce přistupuje mnoho uživatelů, můžete zvýšit souběžnost použitím menší granularity zamykání. Víte-li, že pro vaše aplikace bude lepší nastavit režim uzamčení na možnost řádek, můžete postupovat některým z těchto způsobů:

- V každém příkazu CREATE TABLE a ALTER TABLE použijte klauzuli LOCK MODE ROW.
- Chcete-li, aby tabulky, které v relaci vytvoříte používaly jako režim uzamčení možnost ROW bez nutnosti použití příkazů CREATE TABLE a ALTER TABLE, nastavte proměnnou prostředí **IFX_DEF_TABLE_LOCKMODE** na hodnotu ROW.
- Nastavte konfigurační parametr DEF_TABLE_LOCKMODE na možnost ROW, aby všechny následující tabulky, které v relaci vytvoříte, automaticky používaly jako režim zamčení ROW, bez nutnosti určovat režim příkazy CREATE TABLE a ALTER TABLE.

Změna režimu uzamčení proměnnou prostředí **IFX_DEF_TABLE_LOCKMODE** nebo konfiguračním parametrem DEF_TABLE_LOCKMODE neovlivní existující tabulky. Ty stále používají režim uzamčení nastavený v době jejich vytvoření.

Pokud jste dříve změnili režim uzamčení tabulky na možnost ROW a později provedli příkaz ALTER TABLE pro úpravu jiných atributů (např. přidání sloupce nebo změna velikost oblasti), nemusíte opět určovat režim uzamčení. Ten zůstane nastaven na možnost ROW a není změněn na výchozí možnost PAGE.

Režim uzamčení jednotlivých tabulek můžete potlačit určením režimu uzamčení klauzulí LOCK MODE v příkazech CREATE TABLE a ALTER TABLE. Následující seznam ukazuje pořadí priorit pro režim uzamčení tabulky:

- Výchozí nastavení je zamykání stránek. Databázový server použije výchozí nastavení, pokud nenastavíte konfigurační parametr, proměnnou prostředí a neurčíte klauzuli LOCK MODE v příkazech SQL.
- Pokud nastavíte konfigurační parametr DEF_TABLE_LOCKMODE, databázový server použije tuto hodnotu, pokud nenastavíte proměnnou prostředí a neurčíte klauzuli LOCK MODE v příkazech SQL.
- Pokud nastavíte proměnnou prostředí **IFX_DEF_TABLE_LOCKMODE**, tato hodnota potlačí konfigurační parametr DEF_TABLE_LOCKMODE a výchozí systémové nastavení. Databázový server použije tuto hodnotu pokud neurčíte klauzuli LOCK MODE v příkazechSQL.
- Pokud určíte klauzuli LOCK MODE v přikazu CREATE TABLE nebo ALTER TABLE, tato hodnota potlačí proměnnou prostředí **IFX_DEF_TABLE_LOCKMODE**, konfigurační parametr DEF_TABLE_LOCKMODE a výchozí nastavení.

Čekání na zámek

Výchozí chování databázového serveru v případě, že uživatel narazí na zámek, je vrátit aplikaci chybu. Chcete-li čekat na zámek neomezeně dlouho, proveďte následující příkaz SQL:

```
SET LOCK MODE TO WAIT;
```

Můžete také čekat specifický počet sekund, jak ukazuje tento příklad:

```
SET LOCK MODE TO WAIT 20;
```

Provedením následujícího příkazu se vrátíte k výchozímu chování (nečekání na zámky):

```
SET LOCK MODE TO NOT WAIT;
```

Zamykání příkazem SELECT

Typ a trvání zámků, které databázový server umísťuje, záleží na úrovni izolace nastavené aplikací, režimu databáze (protokolování, bez protokolování nebo ANSI) a na tom, zda je příkaz SELECT v aktualizačním kurzoru. Tyto zámky mohou ovlivnit výkon, protože ovlivňují souběžnost, jak je popsáno v následujících částech.

Úroveň izolace

Počet a trvání zámků umístěných do dat příkazem SELECT záleží na uživatelem nastavené úrovni izolace. Protože ovlivňuje souběžnost, může úroveň izolace ovlivnit celkový výkon.

Předtím, než provedete příkaz SELECT, můžete nastavit úroveň izolace příkazem SET ISOLATION, který je rozšířením standardu ANSI SQL-92 v systému Informix, nebo ANSI/ISO kompatibilním příkazem SET TRANSACTION. Hlavní rozdíl mezi těmito dvěma příkazy spočívá v tom, že příkaz SET ISOLATION má navíc úroveň izolace kurzorová stabilita a narozdíl od příkazu SET TRANSACTION může být proveden v jedné transakci vícekrát. Příkaz SET ISOLATION představuje rozšíření standardu ANSI SQL-92 v systému Informix. Příkaz SET ISOLATION může změnit trvalou úroveň izolace relace.

Úroveň izolace neaktualizované čtení

Úroveň izolace neaktualizované čtení (nebo nepotvrzené čtení ve standartu ANSI) neumisťuje do řádků čtených příkazem SELECT žádné zámky. Proto je vhodná pro dotazy do statických tabulek.

Pokud se provádí aktualizace, používejte neaktualizované čtení s rozvahou. Při použití neaktualizovaného čtení může čtenář přečíst řádek, který ještě nebyl potvrzen, a mohl by být odstraněn nebo změněn při odvolání transakce. Představte si následující případ:

```
Uživatel 1 spustil transakci.
Uživatel 1 vložil řádek A.
Uživatel 2 přečte řádek A.
Uživatel 1 odvolá řádek A.
```

Uživatel 2 přečetl řádek A, který uživatel 1 odvolal o sekundu později. Takže uživatel 2 přečetl řádek, který nebyl v databázi nikdy potvrzen. Nepotvrzená data, která byla odvolána mohou způsobovat v aplikacích problémy.

Protože databázový server neověřuje ani neumisťuje zámky, nabízí úroveň izolace neaktualizované čtení nejlepší výkon ze všech úrovní izolace, nicméně kvůli potencionálním problémům s nepotvrzenými a odvolanými daty používejte úroveň izolace neaktualizované čtení s rozmyslem.

Protože problémy s nepotvrzenými a odvolanými daty se vyskytují jen při použití transakcí, databáze které transakce nepoužívají (a nemají mechanismus povolování transakcí), používají úroveň izolace neaktualizované čtení jako výchozí nastavení. U databází, které nepodporují protokolování, je neaktualizované čtení jedinou dovolenou úrovní izolace.

Úroveň izolace potvrzené čtení

S nastavenou úrovní izolace potvrzené čtení (i ve standartu ANSI označováno jako potvrzené čtení) čtecí zařízení nejprve ověří, zda nejsou na data umístěny zámky, a až poté vrátí řádek. Díky tomu nemůže čtecí zařízení vrátit nepotvrzené řádky.

Ve skutečnosti neumísťuje databázový server při čtení řádků během potvrzeného čtení žádné zámky. Namísto toho ověří, zda řádek existuje ve vnitřní tabulce zámků.

Potvrzené čtení je výchozí úrovní izolace pro databáze s protokolování, pokud není režim protokolu kompatibilní se standardem ANSI. Pro databáze vytvořené s režimem protokolování, který není kompatibilní se standardem ANSI, je nejvhodnější úroveň izolace pro většinu aktivit právě potvrzené čtení. Pro databáze kompatibilní se standardem ANSI je výchozí úroveň izolace opakovatelné čtení.

Způsoby snížení rizika konfliktů při použití úrovně izolace potvrzené čtení: Pokud je úroveň izolace nastavena na potvrzené čtení, mohou zámky držené jinými relacemi zapříčinit selhání SQL operací, pokud aktuální relace nemůže získat zámek nebo pokud databázový server zjistí zablokování. (K zablokování dojde, pokud dva uživatelé umístí zámky a každý z nich čeká na uvolnění zámku patřící druhému uživateli.) Riziko konfliktů při zamykání můžete snížit volitelným klíčovým slovem LAST COMMITTED v příkazu SET ISOLATION COMMITTED READ. Tím řeknete serveru, aby vrátil poslední potvrzenou verzi řádků i navzdory tomu, že souběžně spuštěná relace drží výlučný zámek. Můžete použít klíčové slovo LAST COMMITTED na B-strom a funkční indexy, tabulky podporující protokolování transakcí a tabulky, které nemají zámky na úrovni stránek nebo výlučné zámky. Další informace o příkazu SET ISOLATION můžete nalézt zde *IBM Informix Guide to SQL: Syntax*.

Pokud se relace používající úroveň izolace neaktualizované čtení nebo potvrzené čtení (ve standartu ANSI/ISO nepotvrzené čtení a potvrzené čtení) pokouší číst řádek, do kterého souběžně běžící relace umístila sdílený zámek, můžete nastavením konfiguračního parametru USELASTCOMMITTED říct databázovému serveru, zda má použít poslední potvrzenou verzi dat namísto čekání na uvolnění zámku. Toto platí jen v databázích s protokolováním transakcí. Poslední potvrzená verze dat je ta, která existovala před každým provedením další aktualizace.

Pokud není hodnota konfiguračního parametru USELASTCOMMITTED nebo proměnné prostředí USELASTCOMMITTED nastavena, nebo je nastavena hodnota NONE, relace s úrovní izolace potvrzené čtení čekají na uvolnění všech výlučných zámků, pokud příkaz SET ISOLATION COMMITTED READ LAST COMMITTED neinstruoval databázový server ke čtení poslední potvrzené verze dat. Další informace o konfiguračním parametru USELASTCOMMITTED najdete v kapitole věnované konfiguračním parametrům v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Nastavení konfiguračního parametru USELASTCOMMITTED na úroveň izolace potvrzené čtení může ovlivnit výkon jen v případě výskytu souběžných konfliktních aktualizací. Pokud dojde k výskytu souběžných konfliktních aktualizací, dotazovací výkon záleží na době trvání transakcí. Například čtecí zařízení používající poslední potvrzenou verzi dat bude potřebovat vrátit zpět aktualizovace provedené jinou souběžně běžící transakcí. Tato situace vyžaduje přečtení jednoho nebo více záznamových protokolů, čímž zvýší vstupně-výstupní přenosy, což může ovlivnit výkon.

Úroveň izolace kurzorová stabilita

Úroveň izolace kurzorová stabilita umístí do řádku, který je právě načítán, sdílený zámek. Tím je zajištěno, že žádný jiný uživatel nemůže aktualizovat tento řádek, dokud uživatel nazačne načítat nový řádek.

V příkladě v pseudokódu na Obrázek 8-1, v místě *načíst řádek* uvolní databázový server zámek na předchozím řádku a umístí ho do právě načítaného řádku. V místě *zavřít kurzor* uvolní databázový server zámek z posledního řádku.

```
nastavit úroveň izolace na hodnotu kurzorová stabilita
vytvořit kurzor pro SELECT * FROM customer
otevřít kurzor
dokud jsou další řádky
   načíst řádek
   provést práci
konec bloku dokud
zavřít kurzor
```

Obrázek 8-1. Zámky umístěné kvůli kurzorové stabilitě

Pokud nepoužijete kurzor pro načítání dat, chová se úroveň izolace kurzorová stabilita stejně jako úroveň izolace potvrzené čtení. Nejsou tedy umístěny žádné zámky.

Úroveň izolace opakovatelné čtení

Úroveň izolace opakovatelné čtení (serializovatelnost ve standartu ANSI a opakovatelné čtení ve standartu ANSI) představuje nejpřísnější úroveň izolace. Při použití opakovatelného čtení zamkne databázový server po dobu trvání transakce všechny řádky, které bude načítat, ne jen jeden právě načítaný.

Příklad v pseudokódu, který znázorňuje Obrázek 8-2 na stránce 8-8, ukazuje, kdy databázový server umísťuje a uvolňuje zámky v případě úrovně izolace opakovatelné čtení. V místě *načíst řádek* umístí databázový server zámek na právě načítaný řádek a na každý řádek potřebný pro načtení tohoto řádku. V místě *zavřít kurzor* uvolní databázový server zámek na posledním řádku.

```
nastavit úroveň izolace na opakovatelné čtení
začátek práce
vytvořit kurzor pro SELECT * FROM customer
otevřít kurzor
dokud jsou další řádky
    načíst řádek
    provést práci
konec bloku dokud
zavřít kurzor
potvrdit práci
```

Obrázek 8-2. Zámky umístěné při použití opakovatelného čtení

Opakovatelné čtení je užitečné během jakéhokoli zpracování více řádků, kde se žádný z nich nesmí změnit. Například předpokládejte, že aplikace musí prověřit zůstatky účtů na třech účtech patřících jedné osobě. Aplikace zjistí zůstatek jednoho účtu a poté druhého. Ve stejnou chvíli spustí jiná aplikace transakci, která sníží zůstatek třetího účtu a zvýší zůstatek prvního. Když původní aplikace zjistí zůstatek třetího účtu, jedná se již o sníženou částku, ale změna prvního účtu není původní aplikací zaznamenána.

Při použití potvrzeného čtení nebo kurzorové stability může předchozí scénář nastat, při použití opakovatelného čtení ne. Původní aplikace zamkne až do konce transakce každý účet, který bude načítat, takže pokus druhé aplikace změnit první účet selže (nebo počká, v závislosti na nastavení SET LOCK MODE).

Protože i načtené řádky zůstávají zamčeny, při sekvenčním čtení tabulky může být zamčen velký počet řádků již irelevantních pro výsledek dotazu. Z tohoto důvodu použijte opakovatelné čtení pro tabulky, k nimž může databázový server přistupovat pomocí indexu. Jestliže index existuje a optimalizátor zvolí sekvenční prohledávání, můžete vynutit použití indexu direktivami. Vynucená změna v dotazovací cestě může negativně ovlivnit výkon.

Zamykání neprotokolujících tabulek

Databázový server neumísťuje zámky stránek ani řádků do neprotokolujících tabulek. Zabránit problémům se souběžnostní, když více uživatelů mění neprotokolující tabulku, můžete jednou z následujících metod:

- Po celou dobu transakce zamkněte tabulku ve výlučném režimu.
- Po celou dobu transakce použijte úroveň izolace opakovatelné čtení.

Důležité: Neprotokolující tabulky s přímým přístupem jsou určeny k rychlému zavádění dat. Doporučený postup před změnou dat v tabulce nebo jejím použitím v transakci je změna tabulky na STANDARD.

Aktualizační kurzor

Aktualizační kurzor je speciálním druhem kurzoru, který může aplikace použít, když existuje možnost, že řádek může být aktualizován. Chcete-li použít aktualizační kurzor, proveďte v aplikaci příkaz SELECT FOR UPDATE. Aktualizační kurzor používá *povýšitelné zámky*, to znamená, že databázový server umístí do řádku, který aplikace načítá, aktualizační zámek (ostatní uživatelé mohou řádek stále vidět), ale zámek je změněn na výlučný, když aplikace použije aktualizační kurzor a příkazy UPDATE...WHERE CURRENT OF pro aktualizaci řádku.

V některých případech může databázový server umístit zámky do řádků, které rozpoznal, ale nenačetl. Jestli toto chování nastane, záleží na tom, jak databázový server provádí příkazy jazyka SQL.

Výhoda aktualizačního kurzoru spočívá v tom, že můžete zobrazit řádek, zatímco ho ostatní uživatelé nemohou měnit nebo ho zobrazit s použitím aktualizačního kurzoru, předtím než ho změníte vy.

Pokud řádek neaktualizujete, je výchozím chování databázového serveru uvolnit aktualizační zámek v okamžiku provedení dalšího příkazu FETCH, nebo uzavření kurzoru. Pokud provedete příkaz SET ISOLATION s klauzulí RETAIN UPDATE LOCKS, databázový server do konce transakce neuvolní žádný existující zámek ani neumístí aktualizační zámek.

Příklad v pseudokódu, který znázorňuje Obrázek 8-3, ukazuje, kdy databázový server umisťuje a uvolňuje zámky při použití kurzoru. V místě *načíst řádek 1* umístí databázový server do řádku 1 aktualizační zámek. V místě *načíst řádek 2* uvolní server aktualizační zámek na řádku 1 a umístí aktualizační zámek do řádku 2. Po provedení příkazu SET ISOLATION s klauzulí RETAIN UPDATE LOCKS neuvolní databázový server do konce transakce žádný aktualizační zámek. V místě *načíst řádek 3* umístí aktualizační zámek do řádku 3 a v místě*načíst řádek 4* umístí aktualizační zámek do řádku 4. V místě *potvrdit práci* uvolní server aktualizační zámky z řádků 2, 3 a 4.

```
vytvořit aktualizační kurzor
začátek práce
otevřít kurzor
načíst řádek 1
načíst řádek 2
SET ISOLATION TO COMMITTED READ
    RETAIN UPDATE LOCKS
načíst řádek 3
načíst řádek 4
potvrdit práci
```

Obrázek 8-3. Uvolňování aktualizačních zámků

U databází kompatibilních se standardem ANSI nejsou aktualizační kurzory obvykle potřeba, protože při použití klauzule RETAIN UPDATE LOCKS se každý kurzor chová stejně jako aktualizační kurzor.

Příklad v pseudokódu, který znázorňuje Obrázek 8-4, ukazuje povýšení aktualizačního zámku na výlučný zámek. V místě *načíst řádek* umístí server aktualizační zámek do řádku, jenž bude načítán. V místě *aktualizovat řádek* povýší server zámek na výlučný. V místě *potvrdit práci* server zámek uvolní.

```
vytvořit aktualizační kurzor
začátek práce
otevřít kurzor
načíst řádek
provést práci
aktualizovat řádek (použit příkaz WHERE CURRENT OF)
potvrdit práci
```

Obrázek 8-4. Povyšování aktualizačních zámků

Zámky umístěné při použití příkazů INSERT, UPDATE a DELETE

Při provádění příkazů INSERT, UPDATE nebo DELETE používá databázový server výlučné zámky. Výlučný zámek znamená, že ostatní uživatelé nemohou řádek zobrazit, pokud nepoužijí úroveň izolace neaktualizované čtení. Navíc žádný jiný uživatel nemůže položku před uvolněním zámku aktualizovat ani odstranit.

Kdy databázový server odstraní výlučný zámek závisí na použitém typu protokolování. Když databázový server používá protokolování, odstraní všechny výlučné zámky po dokončení transakce (ať už byla transakce potvrzena nebo odvolána). Když server protokolování nepoužívá, uvolní všechny exkluzivní zámky okamžitě po dokončení příkazů INSERT, UPDATE nebo DELETE.

Sledování a správa zámků

Databázový server ukládá zámky ve vnitřní tabulce zámků. Při čtení řádku databázový server zkontroluje, jestli řádek nebo přidružená stránka, tabulka nebo databáze nejsou uvedeny v tabulce zámků. Pokud v ní uvedeny jsou, musí databázový server zkontrolovat typ zámku. Následující tabulka ukazuje možné typy zámků.

Typ zámku	Popis	Příkaz obvykle umísťující tento typ zámku
S	Sdílený zámek	SELECT
X	Výlučný zámek	INSERT, UPDATE, DELETE
U	Aktualizační zámek	SELECT při použití aktualizačního kurzoru
B	Bajtový zámek	Všechny příkazy aktualizující sloupec typu VARCHAR

Bajtový zámek je generován jen tehdy, zmenšujete-li velikost datové hodnoty ve sloupci typu VARCHAR. Bajtový zámek existuje jen pro přehrání žurnálu a odvolání transakce, takže je vytvořen jen tehdy, pokud pracujete s databází používající protokolování. Bajtové zámky se objeví ve výstupu příkazu **onstat -k** pouze tehdy, pokud používáte zamykání na úrovni řádku, jinak jsou spojeny do zámku stránky.

V tabulce zámků mohou být navíc uloženy *pokusy o zamknutí* stejných typů jako zámky. V některých případech potřebuje uživatel zaregistrovat svůj možný úmysl zamknout položku, aby do ní ostatní uživatelé nemohli umístit zámek.

V závislosti na typu operace a úrovni izolace může databázový server buď pokračovat ve čtení řádku a umístit do něj vlastní zámek, nebo počká na uvolnění zámku (pokud uživatel spustil příkaz SET LOCK MODE TO WAIT). Následující tabulka ukazuje typy zámků, které uživatel může umístit, pokud již jiný uživatel umístil některý typ zámku. Pokud například jeden uživatel umístil na položku výlučný zámek, obdrží jiný uživatel požadující umístění jakéhokoli druhu zámku (výlučný, aktualizační, sdílený) chybu.

	Umístěn zámek X	Umístěn zámek U	Umístěn zámek S
Požaduje zámek X	Ne	Ne	Ano
Požaduje zámek U	Ne	Ne	Ano
Požaduje zámek S	Ne	Ano	Ano

Sledování zámků

Tabulku zámků zobrazíte příkazem **onstat -k**. Obrázek 8-5 ukazuje výstup po zadání příkazu **onstat -k**.

```
Locks
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
300b77d0 0        40074140 0        HDR+S    10002    106        0
300b7828 0        40074140 300b77d0 HDR+S    10197    123        0
300b7854 0        40074140 300b7828 HDR+IX   101e4    0          0
300b78d8 0        40074140 300b7854 HDR+X    101e4    102        0
 4 active, 5000 total, 8192 hash buckets
```

Obrázek 8-5. výstup příkazu onstat -k

V tomto příkladu vkládá uživatel do tabulky řádek. Při tom umístí následující zámky (v tomto pořadí):

- Sdílený zámek do databáze
- Sdílený zámek do řádku v tabulce katalogu **systables**
- Pokus o výlučný zámek do tabulky
- Výlučný zámek do řádku

Tabulku, do které bude zámek umístěn, určíte následujícím příkazem jazyka SQL. Slovo *tblsnum* nahraďte hodnotou v poli **tblsnum** ve výstupu příkazu**onstat -k**.

```
SELECT *
   FROM SYSTABLES
   WHERE HEX(PARTNUM) = "tblsnum";
```

Kde *tblsnum* je modifikovaná hodnota vrácená příkazem **onstat -k**. Vrátí-li například příkaz **onstat -k** hodnotu 10027f, *tbslnum* bude 0x0010027F.

Chcete-li získat informace o aktivní zámcích, dotazujte tabulku **syslocks** v databázi **sysmaster**. Tabulka **syslocks** obsahuje následující sloupce.

Sloupec	Popis
dbsname	Databáze, do které je umístěn zámek
tabname	Jméno tabulky, do které je umístěn zámek
rowidlk	ID řádku, do kterého je umístěn řádek (0 v případě tabulkového zámku)
keynum	Číslo klíče řádku
type	Typ zámku
owner	ID relace vlatníka zámku
waiter	ID relace prvního čekajícího podprocesu

Konfigurace a sledování počtu zámků

Konfigurační parametr LOCKS určuje počáteční velikost vnitřní tabulky zámků.

Jestliže počet zámků přidělených relaci překročí hodnotu nastavenou v konfiguračním parametru LOCKS, zdvojnásobí databázový server velikost tabulky zámků. Pokaždé když dojde k přetečení tabulky zámků (když je větší než hodnota LOCKS), databázový server zdvojnásobí její velikost. Toto se může opakovat maximálně 15krát. Při zdvojnásobení velikosti tabulky zámků nepřidělí server nikdy více než 100 000 zámků. Po patnáctém zdvojnásobení velikosti tabulky zámků ji server už dále nezvětšuje a aplikace požadující zámek obdrží chybu.

Informace jak určit počáteční hodnotu konfiguračního parametru LOCKS naleznete v části “LOCKS” na stránce 4-14.

Chcete-li si sledovat, kolikrát aplikace obdržela chybu nedostatek zámků, podívejte se do pole **ovlock** ve výstupu příkazu **onstat -p**. Podobné informace naleznete i v tabulce **sysprofile** v databázi **sysmaster**. Následující řádky obsahují relevantní statistické údaje.

Řádek	Popis
ovlock	Kolikrát se relace pokusila překročit maximální počet zámků.
lockreqs	Kolikrát relace požádala o zámek.
lockwts	Kolikrát relace čekala na zámek.

Pokaždé, když databázovy server zvětší velikost tabulky zámků, umístí zprávu do protokolu zpráv. Protokol zpráv byste měli pravidelně sledovat a pokud zjistíte, že databázový server zvětšil velikost tabulky zámků, měli byste zvýšit konfigurační parametr LOCKS.

Tabulka zámků může obsahovat až 9 500 000 zámků, což je maximální hodnota parametru LOCKS (8 000 000) plus 15 dynamických přidělení po 100 000 zámcích. Příliš velká tabulka zámků může snižovat výkon. Přestože je algoritmus čtoucí tabulku zámků výkonný, je čtení velké tabulky zámků pokaždé, když databázový server čte řádek, časově nákladné.

Používá-li databázový server neobvykle vysoký počet zámků, můžete si prohlédnout použití zámků jednotlivými aplikacemi tímto způsobem:

1. Příkazem **onstat -u** se můžete přesvědčit, zda některý z uživatelů nepoužívá mimořádně vysoký počet zámků (velké číslo ve sloupci **locks**).
2. Pokud některý uživatel používá velký počet zámků, prohlédněte si příkazy SQL aplikace a uvažte, zda je vhodnější použít tabulkové zámky nebo samostatné zámky řádků a stránek.

Tabulkový zámek je efektivnější než samostatné zámky řádků, ale snižuje souběžnost.

Jedním ze způsobů, jak snížit počet zámků umístěných do tabulky, je změnit způsob zamykání tabulky ze zámků řádků na zámky stránek. Nicméně zámky řádek snižují celkovou souběžnost tabulky, což může ovlivnit výkon.

Počet zámků umístěných do tabulky můžete také snížit zamykáním tabulky ve výlučném režimu.

Sledování čekání a chyb zámků

Pokud aplikace provede příkaz SET LOCK MODE TO WAIT, počká databázový server na uvolnění zámku a nevrací aplikaci chybu. Při neobvykle dlouhém čekání mohou nabýt uživatelé dojmu zatuhnutí aplikace.

Výstup programu **onstat -u**, který znázorňuje Obrázek 8-6 na stránce 8-13, ukazuje, že relace s ID 84 čeká na zámek (L v prvním sloupci pole **Flags**). Vlastníka zámku najdete příkazem **onstat -k**.

```
onstat -u

Userthreads
address  flags   sessid user     tty    wait    tout locks nreads nwrites
40072010 ---P--D 7      informix -         0       0     0     35       75
400723c0 ---P--- 0      informix -         0       0     0      0        0
40072770 ---P--- 1      informix -         0       0     0      0        0
40072b20 ---P--- 2      informix -         0       0     0      0        0
40072ed0 ---P--F 0      informix -         0       0     0      0        0
40073280 ---P--B 8      informix -         0       0     0      0        0
40073630 ---P--- 9      informix -         0       0     0      0        0
400739e0 ---P--D 0      informix -         0       0     0      0        0
40073d90 ---P--- 0      informix -         0       0     0      0        0
 40074140 Y-BP--- 81    lsuto    4  50205788       0     4    106      221
400744f0 --BP--- 83     jsmit    -         0       0     4      0        0
400753b0 ---P--- 86     worth    -         0       0     2      0        0
40075760 L--PR-- 84      jones    3   300b78d8    -1     2      0        0
 13 active, 128 total, 16 maximum concurrent

onstat -k

Locks
address  wtlist   owner    lklist   type    tblsum rowid  key#/bsiz
300b77d0 0        40074140 0        HDR+S   10002  106     0
300b7828 0        40074140 300b77d0 HDR+S   10197  122     0
300b7854 0        40074140 300b7828 HDR+IX  101e4  0       0
 300b78d8 40075760 40074140 300b7854 HDR+X  101e4  100     0
300b7904 0        40075760 0             S   10002  106     0
300b7930 0        40075760 300b7904      S   10197  122     0
 6 active, 5000 total, 8192 hash buckets
```

Obrázek 8-6. Výstup programu onstat -u ukazující použití zámků

Nalezení vlastníka zámku, na který čeká relace s ID 84::

1. Získejte adresu zámku z výstupu příkazu **onstat -u** v poli **wait** (300b78d8).
2. Najděte tuto adresu (300b78d8) ve výstupu příkazu **onstat -k** v poli **Locks address**. Pole **owner** na stejném řádku obsahuje adresu uživatelského procesu (40074140).
3. Najděte tuto adresu (40074140) v poli **Userthreads** ve výstupu programu **onstat -u**. Pole **sessid** na stejném řádku obsahuje ID relace (81) vlastníka zámku.

Řešením tohoto problému je nenásilné ukončení aplikace samotným uživatelem. Pokud toto řešení není možné, můžete zabít proces aplikace nebo odstranit relaci příkazem **onmode -z**.

Sledování zablokování

Zablokování nastává v případě, kdy dva uživatelé umístí zámky a každý z nich čeká na uvolnění zámku druhého uživatele.

Například uživatel **josef** umístil zámek do řádku 10. Uživatelka **jana** umístila zámek do řádku 20. Předpokládejme, že **jana** chce umístit zámek do řádku 10 a **josef** chce umístit zámek do řádku 20. Pokud oba spustí příkaz SET LOCK MODE TO WAIT, mohou na sebe potencionálně čekat navždy.

Systém Informix používá pro zjištění zablokování tabulku zámků a zabrání mu dříve než nastane. Před přidělením zámku prohlédne databázový server seznam zámků všech uživatelů. Pokud uživatel drží zámek na zdroji, který chce žadatel zamknout, databázový server prohledá seznam čekajících zámků tohoto uživatele, aby zjistil, jestli nečeká na zámek který drží žadatel. Jestli ano, žadatel obdrží chybu zablokování.

Chyby zablokování mohou být nevyhnutelné, jestliže aplikace často aktualizuje stejné řádky. Navíc některé aplikace mohou být vždy s jinými ve sporu. Aplikace produkující velký počet zablokování je doporučeno nespuštět ve stejnou dobu. Počet zablokování zjistíte z pole **deadlks** ve výstupu příkazu **onstat -p**.

Protože databázový server neověřuje tabulky zámků jiných databázových serverů, nelze při používání distribuovaných aplikací zjistit zablokování předem. Zablokování lze v tomto případě zabránit použitím konfiguračního parametru DEADLOCK_TIMEOUT. Ten určuje počet sekund, po které bude databázový server čekat na odpověď vzdáleného serveru, než vrátí chybu. Přestože prodlevu mohou způsobit i jiné příčiny než zablokování distribuovaných aplikací, chrání tento mechanismus transakci před nekonečným čekáním.

Počet zablokování distribuovaných aplikací můžete zjistit v poli **dltouts** ve výstupu příkazu **onstat -p**.

Sledování úrovně izolace použité relacemi

Příkazy **onstat -g** a **onstat -g** vypíší seznam úrovní izolace používaných relacemi. Následující tabulka stručně vysvětluje hodnoty ve sloupci **IsoLvl**.

Hodnota	Popis
DR	Neaktualizované čtení
CR	Potvrzené čtení
CS	Kurzorová stabilita
CRU	Potvrzené čtení s RETAIN UPDATE LOCKS
CSU	Kurzorová stabilita s RETAIN UPDATE LOCKS
DRU	Neaktualizované čtení s RETAIN UPDATE LOCKS
LC	Potvrzené čtení, poslední potvrzené
RR	Opakovatelné čtení

Pokud spory o zámky nastávají často, ověřte, zda relace používají vhodnou úroveň izolace.

Zámky inteligentních velkých objektů

Protože sloupce inteligentních velkých objektů jsou obvykle o hodně větší než ostatní v tabulce, mají zámky těchto objektů několik jedinečných vlastností. V této části budou zmíněny tyto:

- Typy zámků inteligentních velkých objektů
- Zamykání rozsahu bajtů
- Povyšování zámků
- Úroveň izolace neaktualizované čtení pro inteligentní velké objekty

Typy zámků inteligentních velkých objektů

Pro zamykání inteligentních velkých objektů používá databázový server jednu z následujících úrovní granularity:

- Oddíl hlavičky bloku sbspace
- Inteligentní velké objekty
- Bajtový rozsah inteligentních velkých objektů

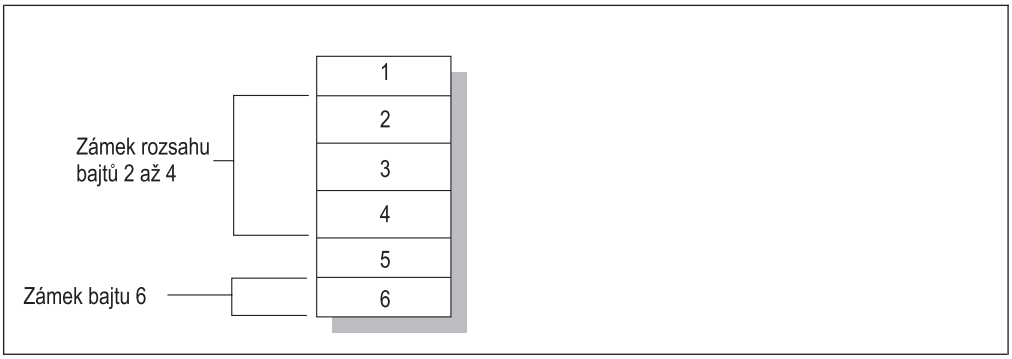
Výchozí granularita zamykání je na úrovni inteligentních velkých objektů. Jinými slovy, když aktualizujete inteligentní velký objekt, při výchozím nastavení zamkne databázový server celý tento inteligentní velký objekty.

Zámek oddílu hlavičky bloku sbspace je vytvořen jen povýšením zámku inteligentního velkého objektu. Další informace naleznete v části "Povyšování zámků" na stránce 8-17.

Zamykání rozsahu bajtů

Namísto zámykání celého inteligentního velkého objektu je možné zamknout jen určitý bajtový rozsah tohoto objektu. Zamykání bajtového rozsahu je výhodnější, protože dovoluje více uživatelům aktualizovat stejný inteligentní velký objekty simultánně (pokud aktualizují rozdílné části). Uživatelé mohou také číst část inteligentního velkého objektu, zatímco jiný uživatel aktualizuje nebo čte jinou část téhož objetku.

Obrázek 8-7 zobrazuje dva zámky umístěné do jednoho inteligentního velkého objektu. První zamyká bajty 2, 3 a 4, druhý jen bajt 6.



Obrázek 8-7. Příklad zamykání bajtového rozsahu

Jak databázový server spravuje zámky bajtového rozsahu

Databázový server spravuje zámky bajtového rozsahu podobně jako ostatní zámky umístěné do řádku, stránky nebo tabulky. Do tabulky zámků musí být v tomto případě uložen i bajtový rozsah.

Pokud uživatel drží zámek bajtového rozsahu a umístí druhý do bajtového rozsahu sousedícího s prvním, sloučí databázový server tyto dva zámky do jednoho přes celý jejich rozsah. Pokud uživatel drží zámky, které znázorňuje Obrázek 8-7, a požádá o zamčení bajtu pět, spojí databázový server zámky umístěné do bajtů dva až šest do jednoho.

Podobně, když uživatel odemkne část bajtů ze zamčeného rozsahu, rozdělí databázový server původní zámek na několik zámků. Obrázek 8-7 znázorňuje situaci, ve které může uživatel odemknout bajt tři, což způsobí rozdělení původního zámku bajtů dva až čtyři na zámek bajtu dva a zámek bajtu čtyři.

Použití zámků bajtového rozsahu

Při výchozím nastavení umísťuje databázový server zámky inteligentních velkých objektů. Chceti-li použít zámky bajtového rozsahu, postupujte následovně:

- Bloku bspace, který obsahuje inteligentní velký objekt, nastavíte zamykání v rozsahu bajtů obslužným programem **onspaces**. Následující příklad nastavuje pro nový blok sbspace zamykání typu bajtový rozsah:

```
onspaces -c -S slo -g 2 -p /ix/9.2/liz/slo -o 0 -s 1000
   -Df LOCK_MODE=RANGE
```

Pokud nastavíte bloku sbspace jako výchozí režim uzamčení bajtový rozsah, zamkne databázový server při aktualizaci inteligentních velkých objektů v tomto bloku jen nezbytně potřebné bajty.

- Otvíraným inteligentním velkým objektům můžete nastavit zamykání bajtového rozsahu pomocí některého z těchto způsobů:

DB-Access

– Ve funkci **mi_lo_open()** DataBlade API nastavte příznak MI_LO_LOCKRANGE.

Konec DB-Access

Jazyk ESQL/C

– Ve funkci **ifx_lo_open()** ESQL/C nastavte příznak LO_LOCKRANGE.

Pokud nastavíte některému inteligentnímu velkému objektu zamykání bajtového rozsahu, databázový server implicitně zamyká při výběru nebo aktualizaci jen potřebné bajty.

Konec Jazyk ESQL/C

- Chcete-li sami určit rozsah zamčených bajtů, použijte některou z následujících funkcí:

DB-Access

– **mi_lo_lock()**

Konec DB-Access

Jazyk ESQL/C

– **ifx_lo_lock()**

Tyto funkce zamknou v inteligentním velkém objektu vámi určený rozsah bajtů. Pokud některou ze zmíněných funkcí umístíte výlučný zámek, příkaz UPDATE neumísťuje při aktualizaci zamčených bajtů do inteligentního velkého objektu zámky.

Databázový server uvolní výlučné zámky bajtového rozsahu umístěné funkcemi **mi_lo_lock()** a **ifx_lo_lock()** na konci transakce. Sdílené zámky bajtového rozsahu umístěné funkcemi **mi_lo_lock()** a **ifx_lo_lock()** uvolní server podle stejných pravidel, jaká platí pro zámky umístěné příkazem SELECT, v závislosti na úrovni izolace. Sdílené zámky bajtového rozsahu můžete uvolnit také těmito funkcemi:

Konec Jazyk ESQL/C

DB-Access

– **mi_lo_unlock()**

Konec DB-Access

Jazyk ESQL/C

– **ifx_lo_unlock()**

Další informace o funkcích DataBlade API naleznete v příručce *IBM Informix DataBlade API Programmer's Guide*. Další informace o funkcích ESQL/C naleznete v příručce *IBM Informix ESQL/C Programmer's Manual*.

Konec Jazyk ESQL/C

Sledování zámků bajtových rozsahů

Pomocí příkazu **onstat -k** lze zobrazit všechny zámky bajtových rozsahů. Pomocí příkazu **onstat -K** lze zobrazit všechny zámky bajtových rozsahů a všechny, kdo na ně čekají. Obrázek 8-8 zobrazuje výstup příkazu **onstat -k**.

```
Byte-Range Locks
rowid/LOid   tblsnum  address  status   owner    offset   size     type
104          200004   a020e90  HDR
[2, 2, 3]             a020ee4  HOLD     a1b46d0  50       10           S
202          200004   a021034  HDR
[2, 2, 5]             a021088  HOLD     a1b51e0  40       5            S
102          200004   a035608  HDR
[2, 2, 1]             a0358fc  HOLD     a1b4148  0        500          S
                      a035758  HOLD     a1b3638  300      100          S
 21 active, 2000 total, 2048 hash buckets
```

Obrázek 8-8. Zámky bajtových rozsahů ve výstupu příkazu onstat -k

Ve výstupu příkazu **onstat -k** lze najít následující informace ohledně zámků bajtových rozsahů.

Sloupec	Popis
rowid	Číslo řádku obsahujícího zamčený inteligentní velký objekt
LOid	Obsahuje tři hodnoty: číslo položky sbspace, číslo bloku a pořadové číslo (hodnota určující pozici v bloku)
tblsnum	Číslo bloku tblspace ve kterém je uložen inteligentní velký objekt
address	Adresa zámku
status	Stav zámku HDR je zástupný symbol. Hodnota HOLD značí, že uživatel uvedený ve sloupci **owner** vlastní zámek. Hodnota WAIT (zobrazovaná jen příkazem **onstat -K**) značí, že uživatel uvedený ve sloupci owner na zámek čeká.
owner	Adresa vlastníka (nebo čekajícího podprocesu) Porovnejte tuto hodnotu s adresou ve výstupu příkazu **onstat -u**.
offset	Pozice zamčeného bajtu vůči počátku inteligentního velkého objektu
size	Počet zamčených bajtů, začínajích na hodnotě uvedené ve sloupci offset
type	S (sdílený zámek) nebo X (výlučný zámek)

Nastavení počtu zámků pro zámykání bajtového rozsahu

Při zamykání bajtového rozsahu může databázový server umístit do jednoho inteligentního velkého objektu více zámků. Počet zámků lze sledovat příkazem **onstat -k**. Přestože se tabulka zámků automaticky zvětšuje, když je zaplněná, můžete chtít zvýšit parametr LOCKS tak, aby odpovídal požadavkům používání zámků a aby sever nemusel tabulce přidělovat další místo dynamicky.

Povyšování zámků

Databázový server používá mechanizmus povyšování zámků pro snížení celkového počtu zámků umístěných do inteligentních velkých objektů. Velké množství zámků může snižovat výkon, protože databázový server kvůli zjištění, zda je objekt zamčen, častěji prohledává tabulku zámků.

Pokud počet zámků držených aplikací překročí 33 procent aktuálně přidělených zámků, pokusí se databázový server povýšit všechny zámky bajtového rozsahu na jediný zámek inteligentního velkého objektu.

Pokud počet zámků inteligentních velkých objektů držených uživatelem (ne počet zámků bajtových rozsahů) představuje 10 nebo více procent kapacity tabulky zámků, pokusí se databázový server povýšit všechny tyto zámky na jediný zámek hlavičky oddílu inteligentního velkého objektu. Tento druh povyšování zámků zvyšuje výkon u aplikací, které aktualizují, nahrávají nebo odstraňují velký počet inteligentních velkých objektů. Například aplikace, odstraňující miliony inteligentních velkých objektů, by bez povyšování zámků zabrala celou tabulka zámků. Algoritmus povyšování zámků má vestavěný mechanismus zabraňující zablokování.

Hlavičku oddílu inteligentních velkých objektů lze určit z výstupu příkazu **onstat -k** podle hodnoty 0 ve sloupci **rowid** a čísla tabulkového prostoru s prvním bajtem a polovinou bajtu vyššího řádu, který odpovídá číslu prostoru dbspace, kde je objekt uložen. Pokud je číslo prostoru tblspace například 0x200004 (hodnoty 0 na nejvyšších řádech byly vypuštěny), je číslo prostoru dbspace 2, což odpovídá číslu prostoru dbspace ve výstupu příkazu **onstat -d**.

Pokus databázového serveru povýšit zámek nemusí být vždy úspěšný. Databázový server například nemůže povýšit zámky bajtového rozsahu na zámek inteligentního velkého objektu, pokud mají jiní uživatelé zámky bajtového rozsahu ve stejném inteligentním velkém objektu. Když databázový server nemůže povýšit zámek bajtového rozsahu, nezmění ho a zpracování pokračuje jako obvykle.

Neaktualizované čtení v inteligentních velkých objektech

Pro inteligentní velké objekty lze použít úroveň izolace neaktualizované čtení. Informace o tom, jak neaktualizované čtení ovlivňuje konzistenci, naleznete v části “Úroveň izolace neaktualizované čtení” na stránce 8-5.

Úroveň izolace neaktualizované čtení pro inteligentní velké objekty nastavíte jedním z těchto způsobů:

- Příkazem SET TRANSACTION MODE nebo příkazem SET ISOLATION.
- Použitím příznaku LO_DIRTY_READ v jedné z těchto funkcí:

DB-Access

– **mi_lo_open()**

Konec DB-Access

Jazyk ESQL/C

– **ifx_lo_open()**

Konec Jazyk ESQL/C

Pokud není konzistence inteligentních velkých objektů důležitá, ale konzistence ostatních sloupců v řádku ano, lze použít úroveň izolace potvrzené čtení, kurzorová stabilita nebo opakovatelné čtení, a inteligentní velké objekty otvírat s příznakem LO_DIRTY_READ.

Kapitola 9. Pokyny k fragmentaci

Obsah kapitoly

Tato kapitola popisuje kritéria výkonu při použití fragmentace tabulky.

Jednou z nejčastějších příčin nízkého výkonu v relačních databázových systémech je soupeření dat, která se nacházejí na jednom zařízení se vstupem - výstupem. Databázový server podporuje fragmentaci tabulek (také *vytváření oddílů*), což uživateli umožňuje uložit data z jediné tabulky na zařízení s několika disky. Správná fragmentace často používaných tabulek může podstatně snížit soupeření vstupu - výstupu.

Další informace o fragmentaci a paralelním provádění popisuje Kapitola 12, “Paralelní databázový dotaz”, na stránce 12-1.

Úvod do problematiky koncepcí a metod fragmentace naleznete v příručce *IBM Informix Database Design and Implementation Guide*. Informace o příkazech jazyka SQL, pomocí kterých je možné spravovat fragmenty, naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Naplánování strategie fragmentace

Strategie fragmentace obsahuje dvě části:

- Schéma distribuce, které udává jak řádky seskupit do fragmentů

  Schéma distribuce můžete určit v klauzuli FRAGMENT BY příkazů CREATE TABLE, CREATE INDEX nebo ALTER FRAGMENT.
- Sada prostorů dbspace (nebo dbslice), ve které můžete vyhledat fragmenty

  U těchto příkazů jazyka SQL můžete v klauzuli IN (volba úložiště) určit sadu prostorů dbspace nebo dbslice.

Postup vytvoření strategie fragmentace:

1. Určete primární cíl fragmentace, který by měl do značné míry záviset na typech aplikací, které k tabulce přistupují.
2. Na základě primárního cíle fragmentace proveďte následující rozhodnutí:
   - Zda fragmentovat data tabulky, index tabulky nebo obojí
   - Jaká je vhodná distribuce řádků nebo indexových klíčů v tabulce
3. Zvolte schéma distribuce založené na výrazu nebo typu cyklická obsluha:
   - Zvolíte-li schéma distribuce založené na výrazu, je nutné navrhnout odpovídající výrazy fragmentů.
   - Zvolíte-li schéma distribuce typu cyklická obsluha, databázový server určí, které řádky mají být vloženy do specifického fragmentu.

   Další informace naleznete v části “Navržení schématu distribuce” na stránce 9-6.
4. Chcete-li dokončit strategii fragmentace, je nutné určit počet a umístění fragmentů:
   - Počet fragmentů závisí na primárním cíli fragmentace.
   - Místo pro umístění fragmentů závisí na počtu dostupných disků v konfiguraci.

Při plánování strategie fragmentace berte v úvahu následující problémy s místem a se stránkami, ke kterým může dojít:

- Ačkoliv se blok o velikosti 4 TB může nacházet na stránce o velikosti 2 kilobajtů, v prostoru dbspace je kvůli omezení formátu identifikátoru rowid možné využít pouze 32 GB.
- U fragmentované tabulky musejí všechny fragmenty používat stejnou velikost stránky.
- U fragmentovaného indexu musejí všechny fragmenty používat stejnou velikost stránky.
- Tabulka může být v jednom prostoru dbspace a index pro tuto tabulku může být v jiném prostoru dbspace. Tyto prostory dbspace nemusí být o stejné velikosti stránky.

Stanovení cílů fragmentace

Pomocí analýzy vaší aplikace a výkonnostní zátěže zhodnoťte rozdíly mezi následujícími cíli fragmentace:

- Zvýšený výkon jednotlivých dotazů

  Chcete-li zvýšit výkony jednotlivých dotazů, fragmentujte tabulky vhodným způsobem a nastavením parametrů zdrojů určete použití systémových zdrojů (paměť, virtuální procesory atd.).
- Snížené soupeření mezi dotazy a transakcemi

- Jestliže databázový server primárně používáte ke zpracovávání transakcí online (OLTP) a pouze někdy k dotazům pro podporu rozhodování, můžete fragmentaci často využít ke snížení soupeření, jestliže souběžně probíhající dotazy vůči tabulce provedou prohledávání indexů a jako výsledek vrátí jen několik řádků.
- Zvýšená dostupnost dat

  Pečlivá fragmentace prostorů dbspace může zlepšit dostupnost dat v případě, že zařízení přestanou fungovat. Fragmenty tabulek na nefunkčním zařízení lze rychle obnovit, přičemž ostatní fragmenty budou stále přístupné.
- Zvýšený výkon zavádění dat

  Pokud pro zavedení tabulky fragmentované na více discích používáte High-Performance Loader (zavaděč HPL), budou jednotkové procesy přiděleny k odlehčenému připojení a data budou souběžně přidělena do fragmentů. Více informací o této metodě zavádění dat naleznete v příručce *IBM Informix High-Performance Loader User's Guide*.

  Pokud chcete rychle přidat data do opravdu velké tabulky, můžete také použít příkaz tabulky alter fragment ON s připojenou klauzulí. Další informace naleznete v části "Zvyšování výkonu připojených a odpojených fragmentů" na stránce 9-17.

Výkon fragmentované tabulky je primárně určován následujícími faktory:

- Použitá volba prostoru pro přiřazování místa na disku fragmentům (popsáno v části "Posouzení faktorů fyzické fragmentace" na stránce 9-5)
- Schéma distribuce použité k přiřazení řádků jednotlivým fragmentům (popsáno v části "Navržení schématu distribuce" na stránce 9-6)

Zvýšení výkonu jednotlivých dotazů

Pokud je primárním cílem fragmentace zvýšit výkon jednotlivých dotazů, pokuste se distribuovat všechny řádky tabulky na různé disky stejnoměrně. Celkový čas pro dokončení dotaze sníží, pokud databázový server nemusí čekat na vyhledávání dat z fragmentu tabulky, který obsahuje více řádků než jiné fragmenty.

Pokud dotazy přistupují k datům pomocí sekvenčního prohledávání podstatné části tabulek, fragmentujte pouze řádky tabulek. Neprovádějte fragmentaci indexu. Pokud dojde k fragmentaci indexu a dotaz musí překračovat hranice fragmentu, aby mohl přistupovat k datům, výkon dotazu může být nižší, než kdybyste fragmentaci neprováděli.

Pokud dotazy přistupují k datům pomocí čtení indexu, můžete zlepšit výkon tím, že použijete stejné schéma distribuce pro index i pro tabulku.

Pokud používáte fragmentaci typu cyklická obsluha, index nefragmentujte. Rozmyslete si, zda index neumístíte do odděleného prostoru dbspace z jiných fragmentů tabulky.

Další informace o zvyšování výkonu dotazů naleznete v části "Výrazy dotazů pro odstraňování fragmentů" na stránce 9-14 a Kapitola 13, "Zvyšování výkonu jednotlivých dotazů", na stránce 13-1.

Snižování soupeření mezi dotazy a transakcemi

Fragmentace může snížit soupeření dat v tabulkách, které používá několik dotazů a aplikací OLTP. Fragmentace často snižuje soupeření, když více současně probíhajících dotazů vůči tabulce provede prohledávání indexu a jako výsledek vrátí jen několik řádků. U tabulek podléhajících tomuto typu zavádění dat proveďte fragmentaci indexových klíčů i řádků s daty, a to pomocí schématu distribuce umožňujícího každému dotazu odstraňovat z vyhledávání nepotřebné fragmenty. Použijte distribuční schéma založené na výrazu. Další informace naleznete v části "Použití schémat distribuce k odstranění fragmentů" na stránce 9-13.

Chcete-li provést fragmentaci tabulky a snížit tak soupeření, nejprve prozkoumejte, do kterých částí tabulky jaké dotazy přistupují. V dalším kroku proveďte fragmentaci dat tak, aby některé z dotazů byly nasměrovány k jednomu fragmentu, zatímco jiné dotazy mohou přistupovat jinému fragmentu. Databázový server provede toto směrování po vyhodnocení pravidla fragmentace pro tabulku. Na závěr uložte fragmenty na oddělené disky.

Úspěšnost snížení soupeření závisí na tom, kolik toho víte o distribuci dat a plánování dotazů v tabulce. Pokud je například distribuce dotazů vůči tabulce nastavena tak, aby byly všechny řádky přístupné zhruba ze stejného stupně, pokuste se řádky distribuovat rovnoměrně mezi fragmenty. Pokud jsou však určité hodnoty přístupné z vyššího stupně než ostatní, je možné tento rozdíl vyrovnat distribucí řádků mezi fragmenty a vyrovnat tak přístupový stupeň. Další informace naleznete v části “Navržení schématu distribuce založeném na výrazu” na stránce 9-8.

Zvýšení dostupnosti dat

Distribuováním fragmentů tabulek a indexů mezi různé disky či zařízení zvyšujete dostupnost dat v případě, že dojde k jejich poruše. Databázový server bude stále umožňovat přístup k fragmentům uloženým na funkčních discích či zařízeních. Tato dostupnost má důležitý dopad na následující typy aplikací:

- Aplikace nevyžadující přístup k nedostupným fragmentům

  Dotaz, který nevyžaduje, aby databázový server přistupoval k datům v nedostupném fragmentu, může stále úspěšně načítat data z dostupných fragmentů. Například pokud výraz distribuce používá jediný sloupec, databázový server může určit, zda je řádek obsažen ve fragmentu, aniž by k němu přistupoval. Pokud dotaz přistupuje pouze k řádkům obsaženým v dostupných fragmentech, může být tento dotaz úspěšný i v případě, že jsou některá data v tabulce nedostupná. Další informace naleznete v části “Navržení schématu distribuce založeném na výrazu” na stránce 9-8.
- Aplikace, které akceptují nedostupnost dat

  Některé aplikace mohou být navrženy tak, aby přijímaly nedostupnost dat ve fragmentu a vyžadovaly schopnost načítat dostupná data. Před provedením dotazu mohu tyto aplikace provedením příkazu SET DATASKIP určit fragmenty, které mají být vynechány. Nebo může administrátor databázového serveru pomocí volby onspaces -f určit, které fragmenty jsou nedostupné.

Pokud je cílem fragmentace zvýšená dostupnost dat, proveďte fragmentaci řádků tabulky i indexových klíčů, aby byla v případě poruchy diskové jednotky některá data stále dostupná. Pokud musejí mít aplikace vždy možnost přistupovat k určité části dat, zachovejte řádky ve stejném zrcadleném prostoru dbspace.

Zvyšování granularity zálohování a obnovení

Při rozhodování o distribuci prostorů dbspace mezi disky berte v úvahu následující faktory týkající se zálohování a obnovení:

- **Dostupnost dat.** Až se budete rozhodovat, kam umístit tabulky nebo fragmenty, nezapomínejte, že pokud dojde k poruše zařízení obsahující prostor dbspace, všechny tabulky nebo fragmenty v tomto prostoru budou nepřístupné, přestože tabulky a fragmenty v jiných prostorech dbspace přístupné jsou. Potřeba omezit nedostupnost dat v případě poruchy disku může mít vliv na tabulky, které v konkrétním prostoru dbspace budete chtít seskupit.
- **Studené a teplé obnovení.** Pokud dojde k poruše prostoru dbspace, který obsahuje kritická data, je nutné provést chladné obnovení, dojde-li však k poruše prostoru dbspace, který obsahuje méně významná data, bude nutné provést teplé obnovení. Potřeba minimalizovat dopad studených obnovení může mít vliv na prostor dbspace, který používáte k ukládání kritických dat.

Více informací o zálohování a obnovení naleznete v příručce *IBM Informix Backup and Restore Guide*.

Kontrola dat a dotazů

Chcete-li určit strategii fragmentace, je nutné vědět, jak jsou data v tabulce používána. Pomocí následujících kroků získáte informace o tabulce, kterou je možné fragmentovat.

Jak získat informace o tabulce:

1. Identifikováním dotazů rozhodujících pro výkon určíte, zda se jedná o dotazy typu zpracovávání transakcí online (OLTP) nebo typu systém podpory rozhodování (DSS).
2. Pomocí příkazu SET EXPLAIN určíte způsob přístupu k datům.

   Další informace o výstupu příkazu SET EXPLAIN naleznete v části "Sestava, která zobrazuje plán dotazů zvolený optimalizátorem" na stránce 10-10. Někdy k určení způsobu přístupu k datům stačí jednoduše zkontrolovat příkazy SELECT spolu se schématem tabulky.
3. Určete, jakou část dat každý dotaz kontroluje.

   Například pokud dochází ke čtení konkrétních řádků v tabulce většinu času, je možné je odloučit do malého fragmentu a tím snížit soupeření vstupu - výstupu u ostatních fragmentů.
4. Určete, pomocí kterých příkazů bude možné vytvářet dočasné soubory.

   Dotazy pro podporu rozhodování typicky vytvářejí a přistupují k dočasným souborům o velké velikosti. Umístění dočasných prostorů dbspace může vážně ovlivnit výkon.
5. Pokud jsou konkrétní tabulky vždy spojeny v dotazu pro podporu rozhodování, rozložte fragmenty těchto tabulek mezi různé disky.
6. Kontrolou sloupců tabulky určíte, které schéma fragmentace umožňuje u dotazů pro podporu rozhodování každý proces prohledávání zaneprázdnit.

   Chcete-li zjistit, jakým způsobem jsou hodnoty sloupců distribuovány, vytvořte distribuci sloupce pomocí příkazu UPDATE STATISTICS a distribuci překontrolujte pomocí obslužného programu **dbschema**.

   ```
   volba dbschema -d databáze -hd tabulka
   ```

Posouzení faktorů fyzické fragmentace

Při fragmentaci tabulky se problematika fyzického umístění týkající se tabulek vztahuje na jejich jednotlivé fragmenty. Podrobnější informace popisuje Kapitola 6, "Úvahy o výkonu tabulek", na stránce 6-1. Protože je každý fragment na disku uložen ve svém vlastním prostoru dbspace, je třeba tyto problémy adresovat odděleně pro fragmenty nacházející se na každém disku.

Fragmentované a nefragmentované tabulky se liší následovně:

- U fragmentované tabulky je každý fragment uložen v odděleném prostoru dbspace, nebo je v rámci jediného prostoru dbspace vytvořeno více oddílů tabulky.

  Nefragmentovaná tabulka může být uložena ve výchozím prostoru dbspace aktuální databáze.

  Bez ohledu na to, zda je tabulka fragmentovaná či nikoli, doporučujeme pro každý prostor dbspace na každém disku vytvořit samostatný blok.
- Velikosti oblasti pro rozšíření bývají u fragmentované tabulky obvykle nižší než u ekvivalentní nefragmentované tabulky, protože se velikost fragmentů nezvyšuje po tak velkých částech jako celá tabulka. Více informací o tom, jak odhadnout velikost prostoru k přidělení naleznete v části "Odhad velikosti tabulky" na stránce 6-6.
- Ve fragmentované tabulce není ukazatel řádku jedinečným a neměnným ukazatelem řádku v disku. Aby bylo možné myší ukázat na řádek, používá databázový server interně uvnitř

indexu kombinaci ID fragmentu a ukazatele řádků. Tato dvě pole jsou jedinečná, ale lze je změnit během existence řádku. Aplikace nemůže přistupovat k ID fragmentu - z tohoto důvodu doporučujeme pro přístup ke specifickému řádku ve fragmentované tabulce používat primární klíče. Další informace naleznete v příručce *IBM Informix Database Design and Implementation Guide*.

- Připojený index nebo index v nefragmentované tabulce používá pro ukazatel řádku 4 bajty. Odpojený index používá pro kombinaci ID fragmentu a ukazatele řádků 8 bajtů prostoru na jednu hodnotu klíče. Více informací o tom, jak odhadnout velikost prostoru pro index, naleznete v části "Odhad indexových stránek" na stránce 7-1. Více informací o připojených a odpojených indexech naleznete v části "Fragmentování indexů" na stránce 9-9.

Dotazy pro podporu rozhodování obvykle vytvářejí a přistupují k dočasným souborům o velké velikosti. Umístění dočasných prostorů dbspace může vážně ovlivnit výkon. Další informace o umisťování dočasných souborů naleznete v části "Rozložení dočasných tabulek a souborů řazení mezi několik disků" na stránce 6-5.

Navržení schématu distribuce

Poté, co se rozhodnete, zda fragmentovat řádky tabulky, indexové klíče nebo oboje a po určení způsobu, jakým mají být řádky nebo klíče mezi fragmenty distribuovány, bude třeba vybrat schéma, které bude k této distribuci implementováno.

Databázový server podporuje následující schémata distribuce:

- **Cyklická obsluha.** Tento typ fragmentace umisťuje řádky jeden po druhém do fragmentů a střídá řady fragmentů tak, aby řádky byly rovnoměrně distribuovány.

  U inteligentních velkých objektů můžete v klauzuli PUT příkazu CREATE TABLE nebo ALTER TABLE určit více prostorů sbspace a distribuovat tak tyto objekty pomocí schématu typu cyklická obsluha - tak bude jejich počet v každém prostoru přibližně stejný.

  Aby mohl databázový server určit fragment, do kterého má být umístěn řádek, používá u příkazů INSERT hashovací funkci pro náhodné číslo. U kurzorů INSERT umisťuje databázový server první řádek do náhodného fragmentu, druhý řádek do následujícího fragmentu atd. Pokud je některý z fragmentů plný, je přeskočen.
- **Fragmentace založená na výrazu.** Tento typ fragmentace umisťuje řádky obsahující uvedené hodnoty do stejného fragmentu. Můžete určit *výraz fragmentace*, který bude definovat podmínky pro přiřazování sady řádků k jednotlivým fragmentům - buď pomocí pravidla rozsahu nebo pomocí nějakého arbitrárního pravidla. Můžete určit *zbývající fragment*, který uchová všechny řádky nesplňující podmínky pro žádný jiný fragment. Tento zbývající fragment však snižuje účinnost schématu distribuce založeném na výrazu.

Zvolení schématu distribuce

Tabulka 9-1 porovnává 3 důležité funkce schémat distribuce typu cyklická obsluha a fragmentace založené na výrazu.

Tabulka 9-1. Porovnání schémat distribuce

Distribuce Schéma	Snadnost vyrovnávání dat	Odstranění fragmentů	Vynechání dat
Cyklická obsluha	Automaticky. Data jsou průběžně vyrovnávána.	Databázový server nemůže odstranit fragmenty.	Není možné určit, zda v případě použití funkce vynechání dat dojde k urovnání integrity transakce. Můžete ji však vložit do tabulky fragmentované pomocí cyklické obsluhy.
Fragmentace založená na výrazu	Vyžaduje znalosti distribuce dat.	Pokud jsou použity výrazy u jednoho nebo dvou sloupců, databázový server může odstranit fragmenty u dotazů, které obsahují výraz rozsahu nebo rovnosti.	Můžete určit, zda při použití funkce vynechání dat dojde k urovnání integrity transakce. Není možné vložit řádky, pokud je příslušný fragment pro tyto řádky nefunkční.

Zvolené schéma distribuce závisí na následujících faktorech:

- Funkce, které uvádí Tabulka 9-1 a které chcete využít
- Zda mají dotazy tendenci prohledávat celou tabulku
- Zda znáte distribuci dat, která má být přidána
- Zda mají aplikace tendenci odstraňovat velké množství řádků
- Zda data v tabulce obíhají

Schéma distribuce typu cyklická obsluha v podstatě zajišťuje nejsnadnější a nejspolehlivější způsob vyrovnávání dat. U tohoto typu distribuce však nejsou k dispozici žádné informace o fragmentu, ve kterém se nachází řádek, a databázový server nemůže odstraňovat fragmenty.

Celkově vzato, cyklická obsluha je správnou volbou pouze v případě, když jsou splněny všechny následující podmínky:

- Dotazy mají tendenci prohledávat celou tabulku.
- Neznáte distribuci dat, která má být přidána.
- Aplikace nemají tendenci odstraňovat velké množství řádků. (Pokud ano, může dojít ke snížení vyrovnání zátěže.)

Fragmentace dat založená na výrazu může být tou nejlepší volbou, pokud je splněna jakákoliv z následujících podmínek:

- Aplikace vyvolává četné množství dotazů pro podporu rozhodování, které prohledávají specifické části tabulky.
- Znáte distribuce dat.
- Plánujete obíhání dat v databázi.

Pokud plánujete pravidelně přidávat a odstraňovat velké množství dat, na základě hodnoty sloupce (např. datum) můžete tento sloupec použít ve schématu distribuce. Poté můžete použít příkazy ALTER FRAGMENT ATTACH a ALTER FRAGMENT DETACH, pomocí kterých umožníte obíhání dat v tabulce.

Příkazy ALTER FRAGMENT ATTACH a DETACH se při závádění a odstraňování objemných dat vyznačují následujícími přednostmi:

- Ke zbývajícím fragmentům tabulky mohou přistupovat ostatní uživatelé. Ostatní uživatelé nemohou přistupovat pouze k připojenému nebo odpojenému fragmentu.

- Pokud zvýšíte výkon, bude provedení příkazu ALTER FRAGMENT ATTACH nebo DETACH mnohem rychlejší, než zavádění objemných dat nebo jejich hromadné odstraňování.

Další informace naleznete v části “Zvyšování výkonu připojených a odpojených fragmentů” na stránce 9-17.

V některých případech může příslušné indexové schéma výkonové problémy konkrétního schématu distribuce obejít. Další informace naleznete v části “Fragmentování indexů” na stránce 9-9.

Navržení schématu distribuce založeném na výrazu

Prvním krokem při navrhování tohoto typu schématu distribuce je určení dat v tabulce, a to zejména distribuce hodnot sloupce, ve kterém chcete založit výraz fragmentace. Chcete-li tyto informace získat, spusťte u tabulky příkaz UPDATE STATISTICS a poté použijte obslužný program **dbschema**, pomocí kterého distribuci zkontrolujete.

Jakmile budete znát distribuci dat, můžete navrhnout pravidlo fragmentace, které bude distribuovat data mezi fragmenty, aby tak bylo dosaženo stanoveného cíle fragmentace. Pokud je primárním cílem zvýšit výkon, výraz fragmentace by měl generovat rovnoměrnou distribuci řádků mezi fragmenty.

Pokud je primárním cílem zvýšit souběžnost, analyzujte dotazy, které přistupují k tabulce. Pokud je k určitým řádkům přistupováno na vyšším stupni než k ostatním, je možné tento stav vyrovnat vybráním nerovnoměrné distribuce dat mezi fragmenty, které vytvoříte.

Nesnažte se používat sloupce, které jsou ve výrazu distribuce často aktualizovány. Tyto aktualizace mohou způsobit, že se řádky přesunou z jednoho fragmentu na jiný (tzn. že jsou z jednoho fragmentu odstraněny a do jiného přidány). Tato aktivita zvyšuje zahlcení procesoru a vstupu - výstupu.

Chcete-li dosáhnout nejúčinnějšího způsobu odstraňování fragmentů, pokuste se vytvořit nepřesahující oblasti založené na jediném sloupci a bez zbývajících fragmentů typu REMAINDER. Databázový server odstraňuje fragmenty z plánů dotazů vždy, když je optimalizátor dotazů schopen určit, zda se hodnoty vybrané pomocí klauzule WHERE nenacházejí ve fragmentech podle pravidla fragmentace založené na výrazu. Tímto pravidlem je možné fragmentům přiřazovat řádky. Další informace naleznete v části “Použití schémat distribuce k odstranění fragmentů” na stránce 9-13.

Návrhy týkající se zlepšení fragmentace

Níže uvedené návrhy jsou návodem pro fragmentaci tabulek a indexů:

- Chcete-li dosáhnout optimálního výkonu dotazů pro podporu rozhodování, pomocí fragmentace tabulky zvýšíte podobnost, neprovádějte však fragmentaci indexů. Odpojte indexy a umístěte je do odděleného prostoru dbspace.
- Chcete-li dosáhnout nejlepšího výkonu v OLTP, soupeření mezi relacemi snížíte pomocí fragmentovaných indexů. Často je možné provést fragmentaci indexu pomocí jeho klíčové hodnoty, což znamená, že dotaz OLTP musí prohlédnout pouze jeden fragment, aby mohl najít umístění řádku.

  Pokud klíčová hodnota nesníží soupeření, když si několik uživatelů současně prohlédne stejnou sadu hodnot (například rozsah dní), zvažte fragmentaci indexu při jiné hodnotě použité v klauzuli WHERE. Chcete-li omezit administraci fragmentů, rozmyslete si, zda budete některé indexy fragmentovat, a to obzvláště v případě, že nebude možné nalézt vhodný výraz fragmentace, pomocí kterého by se soupeření omezilo.

- Až budou dotazy pro podporu rozhodování následně číst tabulku, fragmentujte data pomocí cyklické obsluhy. Pokud chcete data rozložit rovnoměrně mezi disky a pro fragmentaci založenou na výrazu není možné použít žádný sloupec tabulky, fragmentace pomocí cyklické obsluhy je vhodnou metodou. V případě dotazů DSS jsou však čteny všechny fragmenty.
- Chcete-li omezit počet požadovaných prostorů dbspace a snížit čas potřebný pro vyhledávání, můžete v rámci stejného prostoru dbspace vytvořit více oddílů.
- Používáte-li výrazy, vytvořte je tak, aby mezi disky byly spíše než požadavky vstupu - výstupu vyrovnávány počty dat. Pokud například většina dotazů přistupuje pouze k určité části dat v tabulce, nastavte výraz fragmentace tak, aby se aktivní části tabulky rozkládaly na disky, i kdyby toto uspořádání vedlo k nerovnoměrné distribuci řádků.
- Výrazy fragmentace by měly být srozumitelné. Výrazy fragmentace mohou být libovolně složité. U složitých výrazů však déle trvá jejich vyhodnocení. Takovéto výrazy také mohou zabraňovat odstraňování fragmentů z dotazů.
- Uspořádejte výrazy fragmentace tak, aby byla nejrestriktivnější podmínka pro každý prostor dbspace testována nejdříve v rámci výrazu. Databázový server testuje, zda zda hodnota splňuje podmínky daného fragmentu. Jestliže není splněna podmínka pro testy tohoto fragmentu, vyhodnocování se zastaví. Pokud je tedy jako první umístěna taková podmínka, která nejpravděpodobněji nesplňuje podmínky, před přesunem databázového serveru k dalšímu fragmentu je k vyhodnocení potřeba méně podmínek. Například v následujícím výrazu testuje databázový server všech šest podmínek nerovnosti pomocí pokusu o vložení řádku s hodnotou 25:

  ```
  x >= 1 and x <= 10 in dbspace1,
  x > 10 and x <= 20 in dbspace2,
  x > 20 and x <= 30 in dbspace3
  ```

  Během tohoto porovnávání je nutné zkontrolovat pouze čtyři podmínky v následujícím výrazu: první nerovnost pro prostor **dbspace1** (x <= 10), první pro prostor **dbspace2** (x <= 20) a obě podmínky pro prostor **dbspace3**:

  ```
  x <= 10 and x >= 1 in dbspace1,
  x <= 20 and x > 10 in dbspace2,
  x <= 30 and x > 20 in dbspace3
  ```
- Snažte se nepoužívat žádné výrazy vyžadující převod datového typu. Převody typu zvyšují dobu, potřebnou pro převod výrazu. Například datový typ DATE je v rámci porovnávání implicitně převeden na datový typ INTEGER.
- Pokud nechcete způsobit administrační nákladovost, neprovádějte fragmentaci u často se měnících sloupců. Pokud například provádíte fragmentaci u sloupce datum a dojde k odstranění starších řádků, fragment s nejstaršími daty bude mít tendenci se vyprázdnit a fragment s aktuálními daty bude mít tendenci se naplnit. Nakonec budete muset vypustit starý fragment a pro novější pořadí bude nutné přidat fragment nový.
- Neprovádějte fragmentaci každé tabulky. Určete kritické tabulky, ke kterým je přistupováno nejčastěji. V disku umístěte pro tabulku pouze jeden fragment.
- Neprovádějte fragmentaci malých tabulek. Fragmentace malé tabulky v rámci více disků není vhodná, protože dojde k zahlcení všech prohledávacích procesů, které k fragmentům přistupují. Vyrovnejte také počet fragmentů s počtem procesorů ve vašem systému.
- Při definování strategie fragmentování nefragmentované tabulky zkontrolujte velikost další oblasti, abyste tak nepřidělovali každému fragmentu příliš velké množství prostoru na disku.

Fragmentování indexů

Při fragmentaci tabulky jsou přidružené indexy fragmentovány implicitně podle použitého schématu fragmentace. Můžete také použít klauzuli FRAGMENT BY EXPRESSION příkazu CREATE INDEX, pomocí které explicitně provedete fragmentaci indexu jakékoliv tabulky. Každý index fragmentované tabulky zabírá svůj vlastní prostor tblpace s vlastními oblastmi.

Index je možné fragmentovat pomocí následujících strategií:

- Stejná strategie fragmentace jako u tabulky
- Strategie fragmentace odlišná od tabulky

Připojené indexy

Připojený index je takový index, který implicitně postupuje podle strategie fragmentace tabulky (schéma distribuce a sada prostorů dbspace, ve které se fragmenty nacházejí. Databázový server automaticky vytvoří připojený index v případě, že nejdříve provedete fragmentaci tabulky.

Chcete-li vytvořit připojený index, neurčujte v příkazu CREATE INDEX strategii fragmentace ani volbu paměti, jako v následujících příkazech jazyka SQL:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN dbsbspace1,
      (a >=5 AND a < 10) IN dbspace2
      ...
   ;

CREATE INDEX idx1 ON tb1(a);
```

U fragmentovaných tabulek využívajících schéma distribuce založené na výrazu nebo cyklické obsluhy můžete také vytvořit více oddílů tabulky nebo indexu v rámci jediného prostoru dbspace. Tak můžete snížit počet požadovaných prostorů dbspace a zjednodušit tím jejich správu.

Chcete-li vytvořit připojený index s oddíly, začleňte název oddílu do příkazů jazyka SQL (viz níže uvedený příklad):

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
     PARTITION part1 (a >=0 AND a < 5) IN dbs1,
     PARTITION part2 (a >=5 AND a < 10) IN dbs1
          ...
 ;

    CREATE INDEX idx1 ON tb1(a);
```

V příkazech CREATE TABLE, CREATE INDEX a ALTER FRAGMENT ON INDEX můžete místo FRAGMENT BY EXPRESSION použít PARTITION BY EXPRESSION (viz níže uvedený příklad):

```
ALTER FRAGMENT ON INDEX idx1 INIT PARTITION BY EXPRESSION
           PARTITION part1   (a <= 10) IN dbs1,
           PARTITION part2   (a <= 20) IN dbs1,
            PARTITION part3  (a <= 30) IN dbs1;
```

Použitím syntaxe ALTER FRAGMENT změníte fragmentované indexy bez oddílů na indexy s oddíly. Následující syntaxe znázorňuje, jak můžete převést fragmentovaný index na index používající oddíly:

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3
CREATE INDEX ind1 ON t1 (c1) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3

ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
     PARTITION part_1 (c1=10) IN dbs1, PARTITION part_2 (c1=20) IN dbs1,
     PARTITION part_3 (c1=30) IN dbs1,
```

Vytvoření tabulky nebo indexu s oddíly povoluje databázovému serveru vyhledávat rychleji a snižuje požadovaný počet prostorů dbspace, což zvyšuje výkon.

Databázový server fragmentuje připojený index podle stejného schématu distribuce, který je použitý u tabulky, a to použitím takového pravidla pro indexové klíče, které bylo použito u dat tabulky. Následkem toho mají připojené indexy následující fyzické charakteristiky:

- Počet fragmentů indexů je stejný jako počet fragmentů dat.
- Každý připojený fragment se nachází ve stejném prostoru dbspace jako odpovídající data tabulky, ale v odděleném prostoru tblspace.
- Připojený index nebo index v nefragmentované tabulce používá u každé položky indexu pro ukazatel řádku 4 bajty. Více informací o tom, jak odhadnout velikost prostoru pro index, naleznete v části “Odhad indexových stránek” na stránce 7-1.

Odpojené indexy

Odpojený index je index s oddělenou strategií fragmentace, kterou nastavíte explicitně pomocí příkazu CREATE INDEX podobně jako v následujících příkazech jazyka SQL:

```
CREATE TABLE tb1(a int)
      FRAGMENT BY EXPRESSION
         (a <= 10) IN tabdbspc1,
         (a <= 20) IN tabdbspc2,
         (a <= 30) IN tabdbspc3;

CREATE INDEX idx1 ON tb1 (a)
      FRAGMENT BY EXPRESSION
         (a <= 10) IN idxdbspc1,
         (a <= 20) IN idxdbspc2,
         (a <= 30) IN idxdbspc3;
```

Tento příklad ukazuje běžnou strategii fragmentace, pomocí které lze fragmentovat indexy stejným způsobem jako tabulky - pro fragmenty indexu je však nutné specifikovat odlišné prostory dbspace. Tato strategie fragmentace, pomocí které se fragmenty indexu umisťují z tabulky do odlišných prostorů dbspace, může zvýšit výkon takových operací jako např. zálohování, obnovy atd.

Ve výchozím nastavení jsou všechny nové indexy vytvořené příkazem CREATE INDEX v dynamickém serveru odpojeny a uloženy v oddělených tabulkových prostorech, pokud není určena syntaxe IN TABLE.

Chcete-li vytvořit odpojený index s oddíly, začleňte název oddílu do příkazů jazyka SQL (viz níže uvedený příklad):

```
CREATE TABLE tb1(a int)
      FRAGMENT BY EXPRESSION
            PARTITION part1 (a <= 10) IN dbs1,
            PARTITION part2 (a <= 20) IN dbs2,
            PARTITION part3 (a <= 30) IN dbs3;

    CREATE INDEX idx1 ON tb1 (a)
          FRAGMENT BY EXPRESSION
            PARTITION part1   (a <= 10) IN dbs1,
            PARTITION part2   (a <= 20) IN dbs2,
            PARTITION part3   (a <= 30) IN dbs3;
```

V příkazech CREATE TABLE, CREATE INDEX a ALTER FRAGMENT ON INDEX můžete místo klauzule FRAGMENT BY EXPRESSION použít klauzuli PARTITION BY EXPRESSION.

Pokud si nepřejete fragmentovat index, můžete celý index vložit do odděleného prostoru dbspace.

Pomocí výrazu můžete fragmentovat index jakékoliv tabulky. U indexu však není možné explicitně vytvořit fragmentaci založenou na cyklické obsluze. Při fragmentaci tabulky

pomocí cyklické obsluhy doporučujeme v rámci zvýšení výkonu vždy převést všechny indexy spojené s tabulkou na odpojené indexy.

Odpojené indexy se vyznačují těmito fyzickými charakteristikami:

- Každý fragment odpojeného indexu se nachází v odlišném prostoru tblspace odpovídajících dat tabulky. Z tohoto důvodu není možné data a indexové stránky v rámci prostoru tblspace prokládat.
- Fragmenty odpojených indexů mají své vlastní oblasti a *ID prostoru tblspace*. ID prostoru tblspace se také nazývá ID *fragmentu* a *číslo oddílu*. Odpojený index používá pro kombinaci ID fragmentu a ukazatele řádků 8 bajtů prostoru na jednu hodnotu klíče. Více informací o tom, jak odhadnout velikost prostoru pro index, naleznete v části "Odhad indexových stránek" na stránce 7-1.

Databázový server ukládá umístění každého fragmentu tabulky a indexu spolu s dalšími souvisejícími informacemi do tabulky systémového katalogu **sysfragments**. Chcete-li přistoupit k následujícím informacím o fragmentovaných tabulkách a indexech, můžete použít tabulku systémového katalogu **sysfragments** :

- Hodnota v poli **partn** je číslo oddílu nebo ID fragmentu tabulky nebo indexu. Číslo oddílu odpojeného indexu se liší od čísla oddílu odpovídajícího fragmentu tabulky.
- Hodnota v poli **strategy** je schématem distribuce použitým ve strategii fragmentace.

Úplný popis hodnot polí, které obsahuje tato tabulka systémového katalogu **sysfragments**, naleznete v příručce *IBM Informix Guide to SQL: Reference*. Informace o použití tabulky systémového katalogu **sysfragments**, pomocí které je možné sledovat fragmenty, naleznete v části "Monitorování využití fragmentace" na stránce 9-24.

Omezení indexů fragmentovaných tabulek

Pokud databázový server prohledává fragmentovaný index, je třeba prohledat více fragmentů indexů a sloučit výsledky. (Výjimkou je případ, ve kterém je index fragmentovaný podle pravidla určujícího rozsah indexových klíčů a ve kterém prohledávání nepřekročí hranice fragmentu.) Pokud je index fragmentovaný, může následkem tohoto požadavku dojít během prohledávání indexů ke snížení výkonu.

Následkem těchto výkonnostních faktorů uplatňuje databázový server vůči indexům následující omezení:

- Indexy není možné fragmentovat pomocí cyklické obsluhy.
- Jedinečné indexy není možné fragmentovat pomocí výrazu, který obsahuje sloupce, nacházející se v indexovém klíči.

Například následující příkaz není platný:

```
CREATE UNIQUE INDEX ia on tab1(col1)
   FRAGMENT BY EXPRESSION
      col2<10 in dbsp1,
      col2>=10 AND col2<100 in dbsp2,
      col2>100 in dbsp3;
```

Fragmentování dočasných tabulek

Je možné fragmentovat explicitní dočasné tabulky v prostorech dbspace, které se nacházejí v různých discích. Více informací o explicitních a implicitních dočasných tabulkách naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Pomocí klauzule TEMP TABLE příkazu CREATE TABLE můžete vytvořit dočasnou fragmentovanou tabulku. U fragmentovaných dočasných tabulek však není možné změnit

strategii fragmentace (tak jako u trvalých tabulek). Databázový server odstraňuje zároveň dočasnou tabulku i fragmenty pro ni vytvořené.

Pro explicitní dočasnou tabulku můžete definovat vlastní strategii fragmentace nebo můžete strategii nechat dynamicky určit databázovým serverem.

Použití schémat distribuce k odstranění fragmentů

Odstranění fragmentů je funkce databázového serveru, která snižuje počet fragmentů spojených s činností databáze. Tato schopnost může výrazně zvýšit výkon a snížit soupeření u disků, na kterých se nacházejí fragmenty.

Odstranění fragmentů zlepšuje dobu odezvy pro daný dotaz a souběžnost dotazů. Protože databázový server nepotřebuje číst v nadbytečných fragmentech, dochází u dotazu k omezení vstupu - výstupu. Aktivita ve frontách LRU je také omezena.

Použijete-li vhodné schéma distribuce, databázový server může odstranit fragmenty z následujících databázových operací:

- Načítací část příkazů SELECT, INSERT, DELETE nebo UPDATE v jazyku SQL

  Před aktuálním vyhledáváním může databázový server odstranit fragmenty pomocí optimalizovaných příkazů jazyka SQL.
- Spojení vnořených smyček

  Po získání hodnoty klíče z vnější tabulky může databázový server z vnitřní tabulky odstranit fragmenty, které mají být vyhledány.

Zda může databázový server odstraňovat fragmenty z vyhledávání, závisí na dvou faktorech:

- Schéma distribuce ve strategii fragmentace prohledávané tabulky
- Typ výrazu dotazu (výraz v klauzuli WHERE příkazů SELECT, INSERT, DELETE nebo UPDATE)

Výrazy fragmentace pro odstraňování fragmentů

Pokud je strategie fragmentování definována pomocí některého z následujících operátorů, k odstranění fragmentů může dojít u dotazu v tabulce.

```
IN
=
<
>
<=
>=
AND
OR
NOT
MATCH
LIKE
```

Pokud výraz fragmentace používá některý z následujících operátorů, nedojde u dotazů v tabulce k odstranění fragmentů.

```
!=
IS NULL
IS NOT NULL
```

Příklady výrazů fragmentace umožňujících odstranění fragmentů naleznete v části “Účinnost odstraňování fragmentů” na stránce 9-15.

Výrazy dotazů pro odstraňování fragmentů

Výraz dotazu (výraz v klauzuli WHERE) se může skládat z následujících výrazů:

- Jednoduchý výraz
- Nejednoduchý výraz
- Více výrazů

V případě eliminace fragmentů bere databázový server v úvahu pouze jednoduché výrazy nebo více jednoduchých výrazů v kombinaci s určitými operátory.

Jednoduchý výraz se skládá z následujících částí:

sloupec operátor hodnota

Část jednoduchého výrazu	Popis
sloupec	Je jediným názvem sloupce Databázový server podporuje odstraňování fragmentů u všech typů sloupců kromě těch, které jsou definovány datovými typy NCHAR, NVARCHAR, BYTE a TEXT.
operátor	Musí se jednat o operátor rovnosti nebo rozsahu
hodnota	Musí se jednat o literál nebo o hostitelskou proměnnou

Následující příklady představují jednoduché výrazy:

```
name = "Fred"
date < "01/25/2007"
value >= :my_val
```

Následující příklady představují nejednoduché výrazy:

```
unitcost * count > 4500
price <= avg(price)
result + 3 > :limit
```

V případě odstraňování fragmentů bere databázový server v úvahu dva typy jednoduchých výrazů podle operátora:

- Výrazy rozsahu
- Výrazy rovnosti

Výrazy rozsahu v dotazu

Výrazy rozsahu používají následující relační operátory:

```
<, >, <=, >=, !=
```

Databázový server dokáže zvládnout odstranění fragmentů jednoho nebo dvou sloupců pomocí jakékoliv kombinace těchto relačních operátorů v klauzuli WHERE.

Databázový server také může odstranit fragmenty, pokud jsou tyto výrazy rozsahu zkombinovány s následujícími operátory:

```
AND, OR, NOT
IS NULL, IS NOT NULL
MATCH, LIKE
```

V případě, že výraz rozsahu obsahuje operátory MATCH nebo LIKE, může databázový server fragmenty odstranit, pokud řetězec končí zástupným znakem. Následující příklady představují výrazy dotazů, které mohou odstranění fragmentů využít:

```
columna MATCH "ab*"
columna LIKE "ab%" OR columnb LIKE "ab*"
```

Výrazy rovnosti v dotazu

Výrazy rovnosti používají následující operátory rovnosti:

```
=, IN
```

Databázový server dokáže zvládnout odstranění fragmentů jednoho nebo dvou sloupců u dotazů pomocí kombinace těchto operátorů rovnosti v klauzuli WHERE. Databázový server také může odstranit fragmenty, pokud jsou tyto výrazy rovnosti zkombinovány s následujícími operátory:

```
AND, OR
```

Účinnost odstraňování fragmentů

Při fragmentaci tabulky pomocí schématu distribuce typu cyklická obsluha nemůže databázový server fragmenty odstranit. Kromě toho, ne všechna schémata distribuce založená na výrazu vykazují při odstraňování fragmentů stejné reakce.

Tabulka 9-2 shrnuje reakce při odstraňování fragmentů u různých kombinací schémat distribuce založených na výrazu a výrazů dotazů. Oddíly ve fragmentovaných tabulkách nemají na níže uvedené reakce při odstraňování fragmentů vliv.

Tabulka 9-2. Odstranění fragmentů u různých kategorií schémat distribuce založených na výrazu a výrazů dotazů

Typ schématu distribuce založeného na výrazu			
Typ výrazu dotazu (klauzule WHERE)	**Nepřesahující indexy v jediném sloupci**	**Přesahující nebo nesousedící fragmenty v jediném sloupci**	**Nepřesahující fragmenty ve více sloupcích**
Rozsah výraz	Fragmenty je možné odstranit.	Fragmenty není možné odstranit.	Fragmenty není možné odstranit.
Výraz rovnosti	Fragmenty je možné odstranit.	Fragmenty je možné odstranit.	Fragmenty je možné odstranit.

Tabulka 9-2 ukazuje, že schémata distribuce sice odstranění fragmentů umožňují, ale jeho účinnost určuje klauzule WHERE dotazu v otázce.

Zvažte například fragmentování tabulky pomocí následujícího výrazu:

```
FRAGMENT BY EXPRESSION
100 < column_a AND column_b < 0 IN dbsp1,
100 >= column_a AND column_b < 0 IN dbsp2,
column_b >= 0 IN dbsp3
```

Databázový server nemůže odstranit žádné fragmenty z vyhledávání, pokud klauzule WHERE obsahuje následující výraz:

```
column_a = 5 OR column_b = -50
```

Databázový server může na druhou stranu odstranit fragment v prostoru dbspace **dbsp3**, pokud klauzule WHERE obsahuje následující výraz:

```
column_b = -50
```

Kromě toho může databázový server odstranit dva fragmenty v prostorech dbspace **dbsp2** a **dbsp3**, pokud klauzule WHERE obsahuje následující výraz:

```
column_a = 5 AND column_b = -50
```

Oddíly ve fragmentovaných tabulkách nemají na reakce při odstraňování fragmentů vliv.

Následující části popisují schémata distribuce, pomocí kterých lze fragmentovat data a zlepšit tak reakce při odstraňování fragmentů.

Nepřesahující fragmenty v jediném sloupci

Pravidlo fragmentace, které vytváří nepřesahující fragmenty v jednom sloupci, je z hlediska odstraňování fragmentů prioritním pravidlem. Výhodou tohoto typu schématu je, že databázový server může odstranit fragmenty u dotazů obsahujících výrazy rozsahu i u dotazů obsahujících výrazy rovnosti. Při navrhování pravidla fragmentace doporučujeme splňovat tyto podmínky. Obrázek 9-1 znázorňuje příklad tohoto typu pravidla fragmentace.

```
...
FRAGMENT BY EXPRESSION
a<=8 OR a IN (9,10) IN dbsp1,
10<a AND a<=20 IN dbsp2,
a IN (21,22,23) IN dbsp3,
a>23 IN dbsp4;
```

Obrázek 9-1. Příklad nepřesahujících fragmentů v jediném sloupci

Pomocí pravidla rozsahu nebo pomocí arbitrárního pravidla založeného na jediném sloupci můžete vytvořit nepřesahující fragmenty. Je možné použít relační operátory AND, IN, OR nebo BETWEEN. Při použití operátoru BETWEEN buďte opatrní. Při analýze klíčového slova BETWEEN databázovým serverem jsou do této analýzy zahrnuty koncové body určené v rozsahu hodnot. Nepoužívejte ve výrazu klauzuli REMAINDER. Pokud klauzuli REMAINDER použijete, databázový server nebude moci vždy odstranit zbývající fragment.

Přesahující fragmenty v jediném sloupci

Jediným omezením pro tuto kategorii pravidel fragmentace je, že je toto pravidlo založeno na jediném sloupci. Fragmenty mohou přesahovat a nesousedit. Je možné použít jakýkoliv rozsah, funkci MOD nebo arbitrární pravidlo založené na jediném sloupci. Obrázek 9-2 znázorňuje příklad tohoto typu pravidla fragmentace.

```
...
FRAGMENT BY EXPRESSION
a<=8 OR a IN (9,10,21,22,23) IN dbsp1,
a>10 IN dbsp2;
```

Obrázek 9-2. Příklad přesahujících fragmentů v jediném sloupci

Pokud použijete tento typ schématu distribuce, může databázový server fragmenty odstraňovat na základě rovnoměrného vyhledávání, ne však na základě vyhledávání rozsahu. Toto schéma distribuce může být přesto užitečné, protože všechny operace INSERT a mnoho operací UPDATE provádí vyhledávání na základě rovnosti.

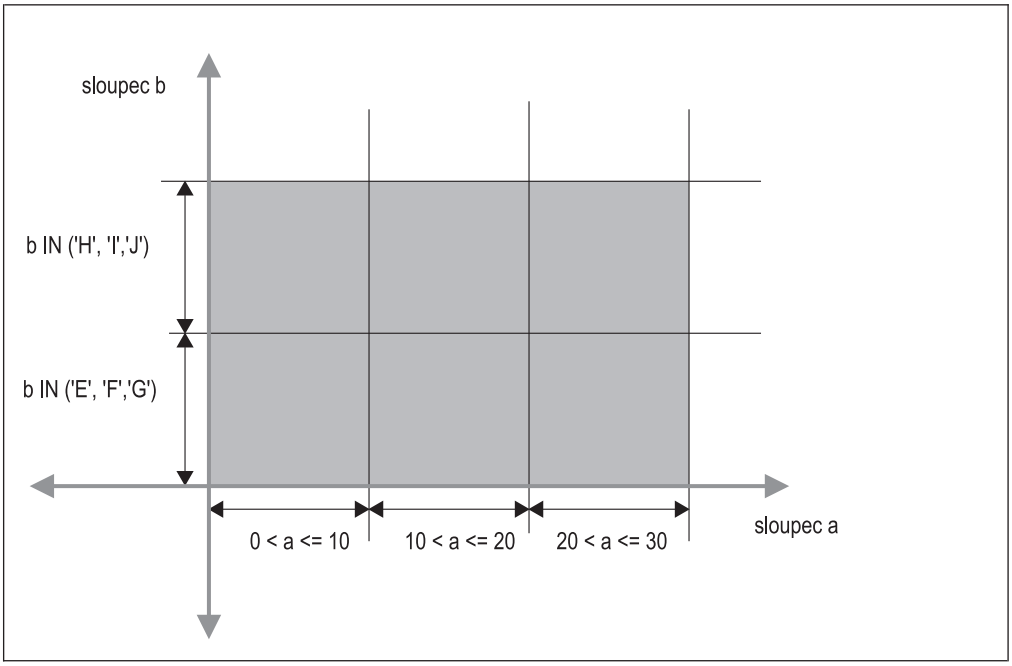
Tato alternativa je přijatelná v případě, že není možné použít výraz, který vytváří nepřesahující fragmenty se sousedícími hodnotami. V případech, kdy tabulka postupem času nabývá na velikosti, budete možná chtít použít pravidlo funkce MOD, pomocí kterého zachováte podobnou velikost fragmentů. Do této kategorie spadají taková schémata distribuce založená na výrazu, která používají pravidla funkce MOD, protože hodnoty v každém fragmentu nesousedí.

Nepřesahující fragmenty, více sloupců

Tato kategorie schématu distribuce založeného na výrazu definuje nepřesahující fragmenty ve více sloupcích pomocí arbitrárního pravidla. Obrázek 9-3 a Obrázek 9-4 znázorňují příklad tohoto typu pravidla fragmentace.

```
...
FRAGMENT BY EXPRESSION
0<a AND a<=10 AND b IN ('E', 'F', 'G') IN dbsp1,
0<a AND a<=10 AND b IN ('H', 'I', 'J') IN dbsp2,
10<a AND a<=20 AND b IN ('E', 'F', 'G') IN dbsp3,
10<a AND a<=20 AND b IN ('H', 'I', 'J') IN dbsp4,
20<a AND a<=30 AND b IN ('E', 'F', 'G') IN dbsp5,
20<a AND a<=30 AND b IN ('H', 'I', 'J') IN dbsp6;
```

Obrázek 9-3. Příklad nepřesahujících fragmentů ve dvou sloupcích



Obrázek 9-4. Schematický příklad nepřesahujících fragmentů ve dvou sloupcích

Pokud použijete tento typ schématu distribuce, může databázový server fragmenty odstraňovat na základě rovnoměrného vyhledávání, ne však na základě vyhledávání rozsahu. Toto schéma distribuce může být přesto užitečné, protože všechny operace INSERT a mnoho operací UPDATE provádí vyhledávání na základě rovnosti. V tomto výrazu nepoužívejte klauzuli REMAINDER. Pokud klauzuli REMAINDER použijete, databázový server nebude moci vždy odstranit zbývající fragment.

Tato alternativa je přijatelná, pokud není možné získat dostatečnou granularitu pomocí výrazu založeného na jediném sloupci.

Zvyšování výkonu připojených a odpojených fragmentů

Mnoho uživatelů pomocí příkazů ALTER FRAGMENT ATTACH a DETACH přidává či odstraňuje velké množství dat ve velmi velkých tabulkách. Pomocí příkazu ALTER FRAGMENT DETACH lze segment dat tabulky rychle odstranit. Stejně tak je možné pomocí příkazu ALTER FRAGMENT ATTACH zavést velké množství dat do existující tabulky

využitím fragmentační technologie. Během vytváření indexů v přežívající tabulce databázovým serverem však může provedení příkazů ALTER FRAGMENT ATTACH a DETACH trvat delší dobu.

Databázový server zajišťuje optimalizaci příkazů ALTER FRAGMENT ATTACH a DETACH, které způsobují, že databázový server opětovně používá indexy v přežívajících tabulkách. Z tohoto důvodu může databázový server odstranit vytváření indexů během operace připojování či odpojování. Důsledkem je:

- Snížení času potřebného k provedení příkazů ALTER FRAGMENT ATTACH a ALTER FRAGMENT DETACH.
- Zvýšení dostupnosti tabulky.

Zvyšování výkonu příkazu ALTER FRAGMENT ATTACH

Chcete-li využít těchto optimalizací výkonu u příkazu ALTER FRAGMENT ATTACH, je nutné splnit všechny níže uvedené požadavky:

- Pro fragmenty tabulky a indexů vytvořte vhodná schémata distribuce.
- Zajistěte, aby následkem fragmentačních výrazů nedocházelo k žádným přesunům dat mezi výslednými oddíly.
- Aktualizujte statistické údaje všech potřebných tabulek.
- Pokud je index v přežívající tabulce jedinečný, u připojených tabulek vytvořte také jedinečné indexy.

Důležité: Ze zvýšení výkonu příkazu ALTER FRAGMENT ATTACH může těžit pouze protokolování databáze. Bez protokolování pracuje databázový server s několika kopiemi stejné tabulky, aby zajistil obnovitelnost dat v případě, že dojde k poruše. Tento požadavek zabraňuje opětovnému použití existujících fragmentů indexu.

Vytvoření vhodných schémat distribuce

Tato část popisuje tři schémata distribuce, která umožňují připojením příkazu ALTER FRAGMENT znovu použít existující indexy:

- Proveďte fragmentaci indexu stejným způsobem jako fragmentaci tabulky.
- Proveďte fragmentaci indexu pomocí stejné sady výrazů fragmentace jako u tabulky.
- Připojením nefragmentovaných tabulek vytvořte fragmentovanou tabulku.

Fragmentace indexu stejným způsobem jako fragmentace tabulky: Vytvořením indexu bez určení strategie fragmentování provedete fragmentaci indexu stejným způsobem jako u tabulky. Strategie fragmentace je schéma distribuce a sada prostorů dbspace, ve kterých se nacházejí fragmenty. Podrobnější informace naleznete v části “Naplánování strategie fragmentace” na stránce 9-2.

Předpokládejme například, že vytvoříte fragmentovanou tabulku a index pomocí následujících příkazů jazyka SQL:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;

CREATE INDEX idx1 ON tb1(a);
```

Dále předpokládejme, že vytvoříte další nefragmentovanou tabulku a později se rozhodnete ji připojit k fragmentované tabulce.

```
CREATE TABLE tb2 (a int, CHECK (a >=10 AND a<15))
   IN db3;
```

```
CREATE INDEX idx2 ON tb2(a)
   IN db3;

ALTER FRAGMENT ON TABLE tb1
      ATTACH
         tb2 AS (a >= 10 and a<15) AFTER db2;
```

Pokud mezi existujícími a novými fragmenty tabulky nedojde k žádnému přesunu dat, může toto připojení využít existujícího indexu **idx2**. Nedojde-li k žádnému přesunu dat:

- Databázový server opětovně použije index **idx2** a převede jej na fragment indexu **idx1**.
- Index **idx1** zůstává jako index se stejnou strategií fragmentace jako tabulka **tb1**.

Pokud databázový server zjistí, že jeden nebo více řádků v tabulce **tb2** patří dřívějším fragmentům tabulky **tb1**, provede tyto operace:

- Vypustí a znovu vytvoří index **idx1**, pomocí kterého začlení řádky, které se původně nacházely v tabulkách **tb1** a **tb2**.
- Vypustí index **idx2**.

Více informací o tom, jak zamezit přesouvání dat mezi existujícími a novými fragmenty tabulky, naleznete v části "Zamezení přesunu dat při připojování fragmentu" na stránce 9-20.

Fragmentace indexu pomocí stejného schématu distribuce jako u tabulky: Vytvořením indexu, který používá stejné výrazy fragmentace jako tabulka, provedete fragmentaci indexu pomocí stejného schématu distribuce jako u tabulky.

Databázový server určuje, zda jsou výrazy fragmentace identické pomocí ekvivalence stromu výrazů, nikoli pomocí algebraické ekvivalence. Prohlédněte si například následující dva výrazy:

```
(col1 >= 5)
(col1 = 5 OR col1 > 5)
```

Ačkoliv jsou tyto dva výrazy algebraicky stejné, nejedná se o identické výrazy.

Předpokládejme, že vytvoříte dvě fragmentované tabulky s indexy pomocí následujících příkazů jazyka SQL:

```
CREATE TABLE tb1 (a INT)
   FRAGMENT BY EXPRESSION
      (a <= 10) IN tabdbspc1,
      (a <= 20) IN tabdbspc2,
      (a <= 30) IN tabdbspc3;
CREATE INDEX idx1 ON tb1 (a)
   FRAGMENT BY EXPRESSION
      (a <= 10) IN idxdbspc1,
      (a <= 20) IN idxdbspc2,
      (a <= 30) IN idxdbspc3;

CREATE TABLE tb2 (a INT CHECK a> 30 AND a<= 40)
   IN tabdbspc4;
CREATE INDEX idx2 ON tb2(a)
   IN idxdbspc4;
```

Dále předpokládejme, že tabulku **tb2** připojíte k tabulce **tb1** pomocí následujícího vzorového příkazu jazyka SQL:

```
ALTER FRAGMENT ON TABLE tb1
   ATTACH tb2 AS (a <= 40);
```

Databázový server může zrušit opětovné vytvoření indexu **idx1** pro toto připojení z následujících důvodů:

- Výraz fragmentace pro index **idx1** je totožný s výrazem fragmentace pro tabulku **tb1**. Databázový server provede následující operace:
  - Rozšíří fragmentaci indexu **idx1** do prostoru dbspace **idxdbspc4**.
  - Převede index **idx2** na fragment indexu **idx1**.
- Protože je kontrolní omezení CHECK totožné s výsledným výrazem fragmentace připojené tabulky, nedojde k žádnému přesunu řádků z jednoho fragmentu do jiného.

  Více informací o tom, jak zamezit přesouvání dat mezi existujícími a novými fragmenty tabulky, naleznete v části "Zamezení přesunu dat při připojování fragmentu" na stránce 9-20.

Připojování nefragmentovaných tabulek: Pokud zkombinujete dvě nefragmentované tabulky do jedné fragmentované, využijete tak zvýšení výkonu operace ALTER FRAGMENT ATTACH.

Předpokládejme například, že vytvoříte dvě nefragmentované tabulky s indexy pomocí následujících příkazů jazyka SQL:

```
CREATE TABLE tb1(a int) IN db1;
   CREATE INDEX idx1 ON tb1(a) in db1;
CREATE TABLE tb2(a int) IN db2;
   CREATE INDEX idx2 ON tb2(a) in db2;
```

Možná budete chtít zkombinovat tyto dvě nefragmentované tabulky pomocí následujícího vzorového schématu distribuce:

```
ALTER FRAGMENT ON TABLE tb1
   ATTACH
      tb1 AS (a <= 100),
      tb2 AS (a > 100);
```

Pokud nedochází k migraci dat mezi fragmenty tabulky **tb1** a **tb2**, databázový server redefinuje index **idx1** pomocí následující strategie fragmentace:

```
CREATE INDEX idx1 ON tb1(a) F
   FRAGMENT BY EXPRESSION
      a <= 100 IN db1,
      a > 100 IN db2;
```

Důležité: Toto chování má za následek odlišnou strategii fragmentace pro index ve verzích databázového serveru starších než verze 7.3 a verze 9.2. Ve starších verzích vytváří příkaz ALTER FRAGMENT ATTACH nefragmentovaný odpojený index v prostoru dbspace **db1**.

Zamezení přesunu dat při připojování fragmentu

Chcete-li předejít přesunu dat, postupujte takto:

Postup zamezení přesunu dat:

1. Vytvořte kontrolní omezení připojené tabulky totožné s výrazem fragmentace, který bude převzat po operaci ALTER FRAGMENT ATTACH.
2. Definujte fragmenty pomocí nepřesahujících výrazů.

Například vytvoříte fragmentovanou tabulku a index pomocí následujících příkazů jazyka SQL:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;

CREATE INDEX idx1 ON tb1(a);
```

Předpokládejme, že vytvoříte další nefragmentovanou tabulku a později se ji rozhodnete připojit k fragmentované tabulce.

```
CREATE TABLE tb2 (a int, check (a >=10 and a<15))
   IN db3;

CREATE INDEX idx2 ON tb2(a)
   IN db3;

ALTER FRAGMENT ON TABLE tb1
   ATTACH
      tb2 AS (a >= 10 AND a<15) AFTER db2;
```

Tato operace ALTER FRAGMENT ATTACH využívá existujícího indexu **idx2**, protože se pomocí následujících kroků názorně zabránilo přesunu dat mezi existujícím a novým fragmentem tabulky:

- Výraz kontrolního omezení v příkazu CREATE TABLE **tb2** je totožný s výrazem fragmentu u tabulky **tb2** v příkazu ALTER FRAGMENT ATTACH.
- Výrazy fragmentu určené v příkazech CREATE TABLE **tb1** a ALTER FRAGMENT ATTACH se nepřekrývají.

Z tohoto důvodu databázový server zachovává index **idx2** v prostoru dbspace **db3** a převádí jej na fragment indexu **idx1**. Index **idx1** zůstává indexem se stejnou strategií fragmentace jako tabulka **tb1**.

Aktualizace statistických údajů o všech použitých tabulkách

Databázový server se pokouší opětovně použít indexy připojených tabulek jako fragmenty výsledného indexu. Odpovídající index připojené tabulky však nemusí existovat nebo může být v důsledku nesprávného diskového formátu nepoužitelný. V těchto případech může být rychlejší vytvořit index připojené tabulky, než vytvářet celý index výsledné tabulky.

Databázový server odhaduje nákladovost vytvoření celého indexu výsledné tabulky. Poté databázový server tuto nákladovost porovnává s nákladovostí vytvoření jednotlivých fragmentů indexu pro připojené tabulky a zvolí nejméně nákladný typ vytváření indexů.

Po úspěšném spuštění příkazu CREATE INDEX (s nebo bez klíčového slova ONLINE) dynamický server automaticky shromáždí následující statistické údaje o nově vytvořeném indexu:

- Statistické údaje na úrovni indexů, ekvivalentní statistickým údajům shromážděným operací UPDATE STATISTICS v režimu LOW, pro všechny typy indexů, včetně B-stromů, rozhraní Virtual Index Interface a funkčních indexů.
- Statistické údaje o distribuci sloupců, které jsou ekvivalentní s distribucí generovanou v operaci UPDATE STATISTICS v režimu HIGH, se vztahují na nekrycí hlavní indexovaný sloupec běžného indexu B-stromu. U velikosti tabulky menší než 1 milion řádků je řešením režimu HIGH verze 1.0. V případě tabulek o vyšší velikosti je řešením verze 0.5. Tabulky obsahující více než 1 milion řádků mají lepší rozlišení, protože obsahují více prostoru pro statistické údaje.

Automaticky shromažďované statistické údaje o distribuci jsou k dispozici optimalizátorovi dotazů při navrhování plánu dotazů pro tabulku, pro kterou byl vytvořen nový index.

Více informací o použití příkazu UPDATE STATISTICS naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Chcete-li zajistit správnost odhadu nákladovostí, doporučujeme před připojením tabulek provést příkaz UPDATE STATISTICS. K získání příslušných statistických údajů postačuje

režim LOW příkazu UPDATE STATISTICS. Optimalizátor pak může pomocí těchto statistických údajů stanovit odhady nákladovosti u opětovně vytvářených indexů.

Příklad situace, ve které neexistuje odpovídající index: Předpokládejme například, že vytvoříte fragmentovanou tabulku a index pomocí následujících příkazů jazyka SQL:

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;
CREATE INDEX idx1 ON tb1(a);
```

Dále předpokládejme, že vytvoříte další dvě nefragmentované tabulky a později se je rozhodnete připojit k fragmentované tabulce.

```
CREATE TABLE tb2 (a int, b int,
   CHECK (a >=10 and a<15)) IN db3;
CREATE INDEX idx2 ON tb2(a) IN db3;
CREATE TABLE tb3 (a int, b int,
   CHECK (a >= 15 and a<20)) IN db4;
CREATE INDEX idx3 ON tb3(b) IN db4;

ALTER FRAGMENT ON TABLE tb1
   ATTACH tb2 AS (a >= 10 and a<15) tb3 AS (a >= 15 and a<20);
```

Tři příkazy CREATE INDEX automaticky vypočítají statistické údaje pro hlavní sloupec každého indexu v režimu HIGH a statistické údaje o indexu a tabulce v režimu LOW.

Jediný případ, kdy je vyžadován příkaz UPDATE STATISTICS LOW FOR TABLE, je po použití příkazu CREATE INDEX v situaci, ve které tabulka obsahuje jiné dřívější indexy (viz následující příklad):

```
CREATE TABLE tb1(col1 int, col2 int);
CREATE INDEX index idx1 on tb1(col1);
  (equivalent to update stats low on table tb1)
LOAD from tb1.unl insert into tb1; (load some data)
CREATE INDEX idx2 on tb1(col2);
```

Příkaz CREATE INDEX idx2 on tb1(col2) se NE ZCELA rovná příkazu UPDATE STATISTICS LOW FOR TABLE tb1, protože příkaz CREATE INDEX u dřívějšího indexu s názvem idx1 neprovádí aktualizaci statistických údajů o úrovních indexů-.

V předcházejícím příkladu tabulka **tb3** neobsahuje index ve sloupci **a**, který může sloužit jako fragment výsledného indexu **idx1**. Databázový server odhadne nákladovost vytvoření fragmentu indexu pro sloupec **a** v používané tabulce **tb3** a porovná tuto nákladovost s opětovným vytvořením celého indexu pro všechny fragmenty ve výsledné tabulce. Databázový server vybere nejméně nákladné vytváření indexů.

Příklad situace, ve které není index tabulky není použitelný: Předpokládejme, že vytvoříte stejné tabulky a indexy jako v předchozí části, ale index ve třetí tabulce bude určovat prostor dbspace, který používá také první tabulka. Následující příkazy jazyka SQL zobrazují tuto možnost:

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;
CREATE INDEX idx1 ON tb1(a);
CREATE TABLE tb2 (a int, b int, check (a >=10 and a<15))
   IN db3;
CREATE INDEX idx2 ON tb2(a)
   IN db3;
```

```
CREATE TABLE tb3 (a int, b int, check (a >= 15 and a<20))
   IN db4;
CREATE INDEX idx3 ON tb3(a)
   IN db2 ;
```

Tento příklad vytvoří index **idx3** tabulky **tb3** v prostoru dbspace **db2**. Následkem toho není index **idx3** použitelný, protože index **idx1** již obsahuje v prostoru dbspace **db2** fragment a strategie fragmentace nepovoluje určit více než jeden fragment v daném prostoru dbspace.

Databázový server opět odhadne nákladovost vytvoření fragmentu indexu pro sloupec **a** v používané tabulce **tb3** a porovná tuto nákladovost s opětovným vytvořením celého indexu **idx1** pro všechny fragmenty ve výsledné tabulce. Databázový server poté vybere nejméně nákladné vytváření indexů.

Zvýšení výkonu příkazu ALTER FRAGMENT DETACH

Chcete-li využít zvýšení výkonu příkazu ALTER FRAGMENT DETACH, vytvořte pro fragmenty tabulky a indexu odpovídající schémata distribuce.

Chcete-li odstranit vytváření indexů během provádění příkazu ALTER FRAGMENT DETACH, použijte jednu z následujících strategií fragmentace:

- Proveďte fragmentaci indexu stejným způsobem jako fragmentaci tabulky.
- Proveďte fragmentaci indexu pomocí stejného schématu distribuce jako u tabulky.

Důležité: Ze zvýšení výkonu příkazu ALTER FRAGMENT DETACH může těžit pouze protokolování databáze. Bez protokolování pracuje databázový server s několika kopiemi stejné tabulky, aby zajistil obnovitelnost dat v případě, že dojde k poruše. Tento požadavek zabraňuje opětovnému použití existujících fragmentů indexu.

Fragmentace indexu stejným způsobem jako fragmentace tabulky

Vytvořením fragmentované tabulky a následně indexu bez určení strategie fragmentování provedete fragmentaci indexu stejným způsobem jako u tabulky.

Předpokládejme například, že vytvoříte fragmentovanou tabulku a index pomocí následujících příkazů jazyka SQL:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2,
      (a >=10 AND a <15) IN db3;
CREATE INDEX idx1 ON tb1(a);
```

Databázový server fragmentuje indexové klíče do prostorů dbspace **db1**, **db2** a **db3** pomocí stejných rozsahů hodnot sloupce **a** jako tabulka, protože příkaz CREATE INDEX neurčuje strategii fragmentace.

Předpokládejme, že se poté rozhodnete odpojit data ve třetím fragmentu pomocí následujícího příkazu jazyka SQL:

```
ALTER FRAGMENT ON TABLE tb1
   DETACH db3 tb3;
```

Protože je u indexu použitá stejná strategie fragmentace jako u tabulky, příkaz ALTER FRAGMENT DETACH po provedení odpojení neprovede opětovné vytvoření indexů. Databázový server vypustí fragment indexu v prostoru dbspace **db3**, aktualizuje tabulky systémového katalogu a odstraní vytváření indexů.

Fragmentace indexu pomocí stejného schématu distribuce jako u tabulky

Vytvořením indexu, který používá stejné výrazy fragmentace jako tabulka, provedete fragmentaci indexu pomocí stejného schématu distribuce jako u tabulky.

Běžnou strategií fragmentace je myšleno fragmetování indexů stejným způsobem jako tabulky - pro fragmenty indexů je však nutné specifikovat odlišné prostory dbspace. Tato strategie fragmentace, pomocí které se fragmenty indexu umisťují z tabulky do odlišných prostorů dbspace, může zvýšit výkon takových operací, jako např. zálohování a obnova.

Předpokládejme například, že vytvoříte fragmentovanou tabulku a index pomocí následujících příkazů jazyka SQL:

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2,
      (a >=10 AND a <15) IN db3;

CREATE INDEX idx1 on tb1(a)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a< 5) IN db4,
      (a >=5 AND a< 10) IN db5,
      (a >=10 AND a<15) IN db6;
```

Předpokládejme, že se poté rozhodnete odpojit data ve třetím fragmentu pomocí následujícího příkazu jazyka SQL:

```
ALTER FRAGMENT ON TABLE tb1
   DETACH db3 tb3;
```

Protože je u indexu použité stejné schéma distribuce jako u tabulky, příkaz ALTER FRAGMENT DETACH po odpojení neprovede opětovné vytvoření indexu. Databázový server vypustí fragment indexu v prostoru dbspace **db3**, aktualizuje tabulky systémového katalogu a odstraní vytváření indexů.

Monitorování využití fragmentace

Po určení strategie fragmentace můžete fragmentaci sledovat následujícími způsoby:

- Spuštěním jednotlivých příkazů obslužného programu **onstat** shromáždíte informace o specifických aspektech probíhajícího dotazu.
- Chcete-li zapsat plán dotazů do výstupního souboru, před spuštěním dotazu proveďte příkaz SET EXPLAIN.

Použití obslužného programu onstat

Chcete-li ověřit zvolenou strategii a určit, zda je mezi fragmenty vyrovnán vstup - výstup, můžete sledovat aktivitu vstupu - výstupu.

Příkaz **onstat -g ppf** zobrazuje počet požadavků čtení a zápisu, odeslaných každému právě otevřenému fragmentu. Protože požadavek může spustit více operací vstupu - výstupu, není z těchto požadavků možné určit, ke kolika jednotlivým diskovým operacím vstupu - výstupu dochází. Z těchto sloupců se však můžete dozvědět o aktivitě vstupu - výstupu.

Samotný výstup však nezobrazuje, ve které tabulce je fragment umístěn. Chcete-li určit tabulku pro fragment, spojte sloupec partnum ve výstupu se sloupcem **partnum** v tabulce systémového katalogu **sysfragments**. Tabulka **sysfragments** zobrazí přidružené **ID tabulky**. Chcete-li pro fragment určit název tabulky, spojte sloupec **table id** v tabulce **sysfragments** se sloupcem **table id** v tabulce **systables**.

Postup zjištění názvu tabulky:

1. Získejte hodnotu v poli **partnum** ve výstupu volby **onstat -g ppf**.
2. Spojením sloupce **tabid** v tabulce systémového katalogu **sysfragments** se sloupcem **tabid** v tabulce systémového katalogu **systables** získáte název tabulky ze **systables**.

   Použijte hodnotu pole **partnum** z kroku 1 v příkazu SELECT.

```
SELECT a.tabname FROM systables a, sysfragments b
   WHERE a.tabid = b.tabid
      AND partn = partnum_value;
```

Použití příkazu SET EXPLAIN

Po fragmentaci tabulky zobrazí výstup příkazu SET EXPLAIN ON, kterou tabulku nebo index databázový server prohledává, aby mohl provést dotaz. Výstup příkazu SET EXPLAIN identifikuje fragmenty pomocí čísla fragmentu. Čísla fragmentů jsou stejná jako čísla ve sloupci **partn** v tabulce systémového katalogu **sysfragments**.

Následující příklad částečného výstupu příkazu SET EXPLAIN zobrazuje dotaz, který využívá odstranění fragmentů a v tabulce **t1** prohledává dva fragmenty:

```
QUERY:
------
SELECT * FROM t1 WHERE c1 > 12

Estimated Cost: 3
Estimated # of Rows Returned: 2

1) informix.t1: SEQUENTIAL SCAN (Serial, fragments: 1, 2)

   Filters: informix.t1.c1 > 12
```

Pokud musí optimalizátor prohledávat všechny fragmenty (tzn. pokud není možné odstranit žádný fragment), výstup příkazu SET EXPLAIN zobrazí fragmenty: ALL. Pokud navíc optimalizátor odstraní všechny fragmenty z výběru (to znamená, že dotazované informace neobsahuje žádný fragment), výstup příkazu SET EXPLAIN zobrazí fragmenty: NONE.

Další informace o tom, jak databázový server odstraňuje fragmenty z výběru, naleznete v části “Použití schémat distribuce k odstranění fragmentů” na stránce 9-13.

Další informace o příkazu SET EXPLAIN ON naleznete v části “Sestava, která zobrazuje plán dotazů zvolený optimalizátorem” na stránce 10-10.

Kapitola 10. Dotazy a optimalizátor dotazů

Obsah kapitoly

Tato kapitola popisuje plány dotazů, vysvětluje způsob, kterým databázový server spravuje optimalizaci dotazů, a podrobně rozebírá faktory, které můžete použít k ovlivnění plánu dotazů. V této kapitole jsou také zahrnuty pokyny týkající se rutiny SPL, mezipamětiuživatelských rutin a spouštěčů.

Funkce PDQ (Parallel database query) databázového serveru poskytují u dotazu nejvyšší potenciální zvýšení výkonu. Kapitola 12, "Paralelní databázový dotaz", na stránce 12-1 popisuje funkci PDQ a správce přidělujícího paměť (MGM) a vysvětluje, jak lze pomocí dotazů řídit použití zdrojů.

Funkce PDQ umožňuje nejvýraznější nárůst výkonu, pokud provedete fragmentaci tabulek tak, jak ji popisuje Kapitola 9, "Pokyny k fragmentaci", na stránce 9-1.

Kapitola 13, "Zvyšování výkonu jednotlivých dotazů", na stránce 13-1 vysvětluje, jak lze zvýšit výkon specifických dotazů.

Plán dotazů

Optimalizátor dotazů vytváří *plán dotazů* pro načtení datových řádků potřebných pro zpracování dotazu.

Optimalizátor musí vyhodnotit různé způsoby, jak lze dotaz provést. Například určí, zda by měly být použity indexy. Pokud dotaz obsahuje spojení, určí optimalizátor jeho postup (hash či vnořená smyčka) a pořadí, ve kterém budou tabulky vyhodnocovány a spojovány. Následující část vysvětluje komponenty plánu dotazů.

Přístupový plán

Způsob, který si optimalizátor zvolí pro čtení tabulky, se nazývá *přístupový plán*. Nejjednodušší metodou přístupu k tabulce je sekvenční čtení, které se označuje jako *prohledávání tabulky*. Optimalizátor zvolí prohledávání tabulek, pokud je nutné číst většinu tabulek nebo pokud tabulky nemají index, který by byl použitelný pro dotaz.

Optimalizátor může také zvolit přístup k tabulce pomocí indexu. Je-li sloupec v indexu stejný jako sloupec ve filtru dotazu, může optimalizátor použít index k vyhledání pouze těch řádků, které dotaz vyžaduje. Optimalizátor může použít *prohledávání pouze klíčů indexu*, pokud jsou požadované sloupce v rámci jednoho indexu v tabulce. Databázový server vyhledá potřebná data z indexu a nepřistupuje k přidružené tabulce.

Důležité: Optimalizátor pro sloupec VARCHAR nezvolí prohledávání pouze klíčů. Pokud chcete využít výhod prohledávání pouze klíčů, změňte pomocí příkazu ALTER TABLE s klauzulí MODIFY sloupec na datový typ CHAR.

Optimalizátor porovná nákladovost každého plánu, a určí tak nejlepší plán. Databázový server odvozuje nákladovost z počtu požadovaných operací vstupu - výstupu, z počtu výpočtů nutných k dosažení výsledků, z počtu řádků, ke kterým se bude přistupovat, počtu řazení a tak dále.

Plán spojení

Pokud dotaz obsahuje více než jednu tabulku, databázový server tyto tabulky v dotazu spojí pomocí filtrů. Například v následujícím dotazu jsou tabulky customer (zákazník) a orders (objednávky) spojena pomocí filtru customer.customer_num = orders.customer_num:

```
SELECT * from customer, orders
WHERE customer.customer_num = orders.customer_num
AND customer.lname = "Higgins";
```

Způsob, který si optimalizátor zvolí pro spojení tabulek, je *plán spojení*. Metodou spojení může být spojení typu vnořená smyčka nebo spojení typu hash.

Kvůli charakteru spojení typu hash může aplikace s úrovní izolace nastavenou na opakovatelné čtení uzamknout všechny záznamy v tabulkách, které jsou zahrnuty do spojení, včetně záznamů, kterým se nezdaří kvalifikovat spojení. Tato situace vede ke snížení souběžnosti mezi připojeními. Naopak spojení typu vnořená smyčka zamkne méně záznamů, avšak při přístupu k velkému množství řádků poskytuje snížený výkon. Znamená to tedy, že každá metoda spojení má své výhody a nevýhody.

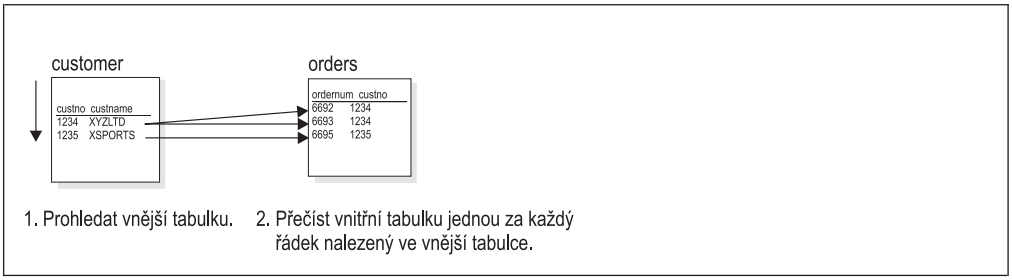
Spojení typu vnořená smyčka

V případě spojení typu vnořená smyčka prohledává databázový server první, neboli *vnější tabulku*, a potom spojí všechny řádky, které projdou filtry tabulky, s řádky nalezenými v druhé, neboli *vnitřní tabulce.* Obrázek 10-1 na stránce 10-3 uvádí tabulky a řádky a pořadí, ve kterém jsou čteny, pro dotaz:

```
SELECT * FROM customer, orders
WHERE customer.customer_num=orders.customer_num
AND order_date>"01/01/2007";
```

Databázový server přistupuje k vnější tabulce prostřednictvím indexu nebo prohledávání tabulek. Databázový server použije nejdříve všechny filtry tabulky. Pro každý řádek, který vyhovuje filtrům vnější tabulky, přečte databázový server vnitřní tabulku a hledá shodu.

Databázový server čte jedenkrát vnitřní tabulku pro každý řádek vnější tabulky, který odpovídá filtrům tabulky. Kvůli potenciálně velkému množství opakování čtení vnitřní tabulky, přistupuje obvykle databázový server k vnitřní tabulce prostřednictvím indexu.



Obrázek 10-1. Spojení typu vnořená smyčka.

Pokud vnitřní tabulka nemá index, může databázový server vytvořit *automatický index* v době provádění dotazu. Optimalizátor může určit, že nákladovost vytvoření *automatického indexu* v době provedení dotazu je menší než nákladovost prohledávání vnitřní tabulky pro každý odpovídající řádek ve vnější tabulce.

Pokud optimalizátor změní poddotaz na spojení typu vnořená smyčka, může použít variantu spojení typu vnořená smyčka nazývanou *smíšené spojení*. Ve smíšeném spojení čte databázový server vnitřní tabulku pouze do té doby, dokud nenalezne shodu. Jinými slovy,

vnitřní tabulka přispívá každému řádku ve vnější tabulce nejvýše jedním řádkem. Více informací o tom, jak optimalizátor pracuje s poddotazy, naleznete v části “Plány dotazů pro poddotazy” na stránce 10-14.

Spojení typu hash

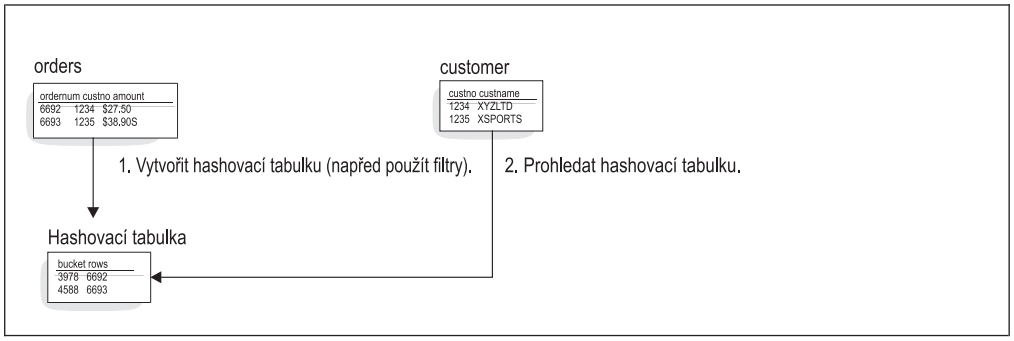
Optimalizátor obvykle používá spojení typu hash, když alespoň jedna ze dvou tabulek spojení nemá index ve sloupci spojení nebo když musí databázový server číst velké množství řádků z obou tabulek. V případě, že databázový server provádí spojení typu hash, není nutný žádný index ani žádné řazení.

Spojení typu hash se skládá ze dvou aktivit: nejprve se vytvoří hashovací tabulka (fáze *tvoření*) a potom proběhne zjišťování této hashovací tabulky (fáze *zjišťování*). Spojení typu hash podrobně zobrazuje Obrázek 10-2.

Ve fázi vytváření čte databázový server jednu tabulku a potom po použití všech filtrů vytvoří hashovací tabulku. Koncepčně je možné o hashovací tabulce uvažovat jako o sérii *sektorů*, z nichž každý má adresu, která je odvozena z hodnoty klíče použitím hashovací funkce. Databázový server neřadí klíče v konkrétním sektoru hashovací tabulky.

Menší hashovací tabulky se mohou vejít do virtuální části sdílené paměti databázového serveru. Databázový server ukládá větší hashovací soubory na disk do prostoru dbspace určeného konfiguračním parametrem DBSPACETEMP nebo proměnnou prostředí **DBSPACETEMP**.

Ve fázi zjišťování čte databázový server ostatní tabulky ve spojení a používá libovolné filtry. Pro každý řádek, který vyhovuje filtrům v tabulce, databázový server použije na klíč hashovací funkci a zjišťuje v hashovací tabulce shodu.



Obrázek 10-2. Provedení spojení typu hash.

Pořadí spojení

Pořadí tabulek, které jsou spojeny v dotazu, je mimořádně důležité. Špatné pořadí spojení může způsobit značný pokles výkonu dotazu.

Následující příkaz SELECT vyvolá třícestné spojení:

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
```

Optimalizátor si může zvolit jedno z následujících pořadí spojení:

- Spojit zákazníka (**customer**) s objednávkami (**orders**). Spojit výsledky s položkami (**items**).
- Spojit objednávky (**orders**) se zákazníkem (**customer**). Spojit výsledky s položkami (**items**).

- Spojit zákazníka (**customer**) s položkami (**items**). Spojit výsledky s objednávkami (**orders**).
- Spojit položky (**items**) se zákazníkem (**customer**). Spojit výsledky s objednávkami (**orders**).
- Spojit objednávky (**orders**) s položkami (**items**). Spojit výsledky se zákazníkem (**customer**).
- Spojit položky (**items**) s objednávkami (**orders**). Spojit výsledky se zákazníkem (**customer**).

Příklad způsobu, jakým databázový server provádí plán podle určitého pořadí spojení, naleznete v části “Příklad provedení plánu dotazů” na stránce 10-5.

Příklad provedení plánu dotazů

Následující příkaz SELECT vyvolá třícestné spojení:

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
```

Předpokládejme také, že v žádné ze třech tabulek nejsou žádné indexy. Předpokládejme, že si optimalizátor zvolí cestu **customer-orders-items** a spojení typu vnořená smyčka pro obě spojení (ve skutečnosti si optimalizátor obvykle zvolí pro dvě tabulky bez indexů ve sloupcích spojení spojení typu hash). Obrázek 10-3 znázorňuje *plán dotazů* vyjádřený v pseudokódu. Další informace týkající se interpretace plánu dotazů naleznete v části “Sestava, která zobrazuje plán dotazů zvolený optimalizátorem” na stránce 10-10.

```
for each row in the customer table do:
   read the row into C
   for each row in the orders table do:
      read the row into O
      if O.customer_num = C.customer_num then
         for each row in the items table do:
            read the row into I
            if I.order_num = O.order_num then
               accept the row and send to user
            end if
         end for
      end if
   end for
end for
```

Obrázek 10-3. Plán dotazů v pseudokódu.

Tato procedura přečte následující řádky:

- Všechny řádky tabulky **customer** jedenkrát.
- Všechny řádky tabulky **orders** jedenkrát pro každý řádek tabulky **customer**.
- Všechny řádky tabulky **items** jedenkrát pro každý řádek páru **customer-orders**.

Tento příklad nepopisuje jediný možný plán dotazů. Jiný plán zcela obrátí úlohy tabulek **customer** a **orders**: pro každou řádku tabulky **orders** přečte všechny řádky tabulky **customer** a hledá shodnou hodnotu **customer_num**. Přečte stejný počet řádků v odlišném pořadí a vytvoří stejnou sadu řádků v odlišném pořadí. V tomto příkladu neexistuje mezi těmito dvěma plány dotazů rozdíl v objemu nutné práce.

Spojení s filtry sloupců

Přítomnost *filtru sloupce* mění situaci. Filtr sloupce je výraz WHERE, který snižuje počet řádků, jimiž tabulka přispívá do spojení. Následující příklad uvádí předchozí dotaz s přidáním filtru:

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
      AND O.paid_date IS NULL
```

Výraz O.paid_date IS NULL vyfiltruje některé řádky, a sníží tak počet řádků, které se použijí z tabulky **orders**. Zvažte plán, který začíná čtením tabulky **orders**. Obrázek 10-4 znázorňuje tento vzorový plán v pseudokódu.

```
for each row in the orders table do:
   read the row into O
   if O.paid_date is null then
      for each row in the customer table do:
         read the row into C
         if O.customer_num = C.customer_num then
            for each row in the items table do:
               read the row into I
               if I.order_num = O.order_num then
                  accept row and return to user
               end if
            end for
         end if
      end for
   end if
end for
```

Obrázek 10-4. Plán dotazů, který používá sloupec filtru.

Nechť hodnota *pdnull* představuje počet řádků v tabulce **orders**, které prošly filtrem. Je to hodnota klauzule **COUNT(*)**, která je výsledkem následujícího dotazu:

```
SELECT COUNT(*) FROM orders WHERE paid_date IS NULL
```

Pokud pro každou objednávku (tabulka order) existuje jeden zákazník (tabulka customer), bude plán, který znázorňuje Obrázek 10-4, číst následující řádky:

- Všechny řádky tabulky **orders** jedenkrát.
- Všechny řádky tabulky **customer** *pdnull*krát.
- Všechny řádky tabulky **items** *pdnull*krát.

Obrázek 10-5 znázorňuje alternativní provedení plánu, který čte nejdříve z tabulky **customer**.

```
for each row in the customer table do:
   read the row into C
   for each row in the orders table do:
      read the row into O
      if O.paid_date is null and
           O.customer_num = C.customer_num then
         for each row in the items table do:
            read the row into I
            if I.order_num = O.order_num then
               accept row and return to user
            end if
         end for
      end if
   end for
```

Obrázek 10-5. Alternativní plán dotazů v pseudokódu.

Jelikož v prvním kroku, který znázorňuje Obrázek 10-5, nebyl použit filtr, čte tento plán následující řádky:

- Všechny řádky tabulky **customer** jedenkrát.
- Všechny řádky tabulku **orders** jedenkrát pro každý řádek z tabulky **customer**.
- Všechny řádky tabulky **items** *pdnull*krát.

Plány dotazů, které znázorňuje Obrázek 10-4 a Obrázek 10-5, vytvoří stejný výstup v odlišné posloupnosti. Liší se v tom, že první plán čte tabulku *pdnull*krát a druhý čte tabulku SELECT COUNT(*) FROM customerkrát. Zvolením vhodného plánu může optimalizátor v reálné aplikaci ušetřit tisíce přístupů na disk.

Spojení s indexy

Předchozí příklady nepoužívají indexy ani omezení. Přítomnost indexů a omezení poskytuje optimalizátoru volby, které mohou výrazným způsobem snížit čas provedení dotazu. Obrázek 10-6 znázorňuje osnovu konstrukce plánu dotazů pro předchozí dotaz při použití indexů.

```
for each row in the customer table do:
   read the row into C
   look up C.customer_num in index on orders.customer_num
   for each matching row in the orders index do:
      read the table row for O
      if O.paid_date is null then
         look up O.order_num in index on items.order_num
         for each matching row in the items index do:
            read the row for I
            construct output row and return to user
         end for
      end if
   end for
end for
```

Obrázek 10-6. Plán dotazů s indexy.

Klíče v indexu jsou seřazeny tak, že když databázový server nalezne první shodnou položku, může číst všechny ostatní řádky s identickými klíči bez dalšího hledání, neboť jsou umístěny ve fyzicky sousedních pozicích. Tento plán čte pouze následující řádky:

- Všechny řádky tabulky **customer** jedenkrát.
- Všechny řádky tabulky **orders** jedenkrát (protože každá objednávka (order) je přidružena pouze jednomu zákazníkovi (customer)).
- Pouze ty řádky tabulky **items**, které se shodují s řádky *pdnull* z páru **customer-orders**.

Tento plán dotazů dosahuje značného snížení nákladovosti v porovnání s plány, které nepoužívají indexy. Inverzní plán, který čte nejdříve tabulku **orders** a v tabulce **customer** vyhledává řádky podle jejich indexů, je ze stejného důvodu rovněž vhodný.

Fyzické pořadí řádků v tabulce také ovlivňuje nákladovost použití indexů. Režie přístupů k vícenásobným řádkům tabulky v pořadí indexů se sníží do stupně, do jakého je tabulka uspořádána vzhledem k indexu. Jsou-li například řádky tabulky **orders** fyzicky řazeny podle čísla zákazníka, bude se vícenásobné vyhledávání objednávek pro daného zákazníka zpracovávat rychleji, než pokud by byla tabulka seřazena náhodně.

V některých případech může použití indexu způsobit další nákladovost. Další informace naleznete v části “Nákladovost vyhledávání indexu” na stránce 10-25.

Plány dotazů zahrnující cestu k indexu spojení typu self-join

Index spojení typu self-join je typem prohledávání indexů, které si můžete představit jako sjednocení mnoha malých prohledávání indexů, z nichž každé má jednu jedinečnou kombinaci sloupců s hlavním klíčem a filtrů u sloupců jiných, než jsou sloupce s hlavním klíčem. Výsledkem sjednocení prohledávání malých indexů je přístupová cesta, která používá pouze podmnožinu úplného rozsahu složeného indexu. Tabulka je logicky spojena sama se sebou a pro každou jedinečnou kombinaci hodnot hlavního klíče se jako filtry svázané s indexem použijí více výběrové klíče indexu, které nejsou hlavní .

Spojení typu self-join je přínosné v situacích, ve kterých:

- Hlavní klíč indexu má mnoho duplikátů.
- Predikáty hlavního klíče nejsou selektivní, avšak predikáty jiných než hlavních klíčů indexu jsou selektivní.

Dotaz, který znázorňuje Obrázek 10-7, obsahuje výstup příkazu SET EXPLAIN pro plán dotazů zahrnující cestu spojení typu self-join.

```
QUERY:
------
SELECT a.c1,a.c2,a.c3 FROM tab1 a WHERE (a.c3 >= 100103) AND
     (a.c3 <= 100104) AND (a.c1 >= 'PICKED      ') AND
     (a.c1 <= 'RGA2        ') AND (a.c2 >= 1) AND (a.c2 <= 7)
     ORDER BY 1, 2, 3

Estimated Cost: 155
Estimated # of Rows Returned: 1
  1) informix.a: INDEX PATH
    (1) Index Keys: c1 c2 c3 c4 c5   (Key-Only)  (Serial, fragments: ALL)
        Index Self Join Keys (c1 c2 )
          Lower bound: informix.a.c1 >= 'PICKED      ' AND (informix.a.c2 >= 1 )
          Upper bound: informix.a.c1 <= 'RGA2        ' AND (informix.a.c2 <= 7 )
        Lower Index Filter: (informix.a.c1 = informix.a.c1 AND
          informix.a.c2 = informix.a.c2 ) AND informix.a.c3 >= 100103
        Upper Index Filter: informix.a.c3 <= 100104
        Index Key Filters:  (informix.a.c2 <= 7 ) AND
                            (informix.a.c2 >= 1 )
```

Obrázek 10-7. Výstup příkazu SET EXPLAIN pro dotaz s cestou k indexu spojení typu self-join.

Obrázek 10-7 znázorňuje, že ve sloupcích c1, c2, c3, c4 a c5 existuje index. Optimalizátor si zvolí jako hlavní klíče sloupce c1 a c2, z čeho vyplývá, že sloupce c1 a c2 mají mnoho duplikátů. Sloupec c3 má méně duplikátů a proto predikáty u sloupce c3 (c3 >= 100103 a c3 <= 100104) mají dobrou selektivitu.

Jak znázorňuje Obrázek 10-7 na stránce 10-8, cesta k indexu spojení typu self-join je spojením typu self-join dvou prohledávání indexů, které používají stejný index. První prohledávání indexů vyhledá každou jedinečnou hodnotu pro sloupce s hlavním klíčem, což jsou sloupce c1 a c2. Jedinečná hodnota sloupce c1 a c2 se potom použije ke zjištění druhého prohledávání indexů, které také použije predikáty u sloupce c3. Díky tomu, že predikáty u sloupce c3 mají dobrou selektivitu, platí tyto skutečnosti:

- Prohledávání indexů na vnitřní straně spojení typu vnořená smyčka je velmi efektivní, neboť vyhledává pouze několik řádků, které vyhovují predikátům sloupce c3.
- Prohledávání indexů nevyhledá řádky navíc.

Z tohoto důvodu pro každou jedinečnou hodnotu c1 a c2 dochází k efektivnímu prohledávání indexů ve sloupcích c1, c2 a c3.

Následující řádky v příkladu ukazují, že si optimalizátor pro tuto tabulku zvolil cestu k indexu spojení typu self-join se sloupci c1 a c2, které jsou pro cestu k indexu spojení typu self-join hlavními klíči:

```
Index Self Join Keys (c1 c2 )
          Lower bound: informix.a.c1 >= 'PICKED      ' AND (informix.a.c2 >= 1 )
          Upper bound: informix.a.c1 <= 'RGA2        ' AND (informix.a.c2 <= 7 )
```

Příklad ukazuje vazby pro sloupec c1 a c2, které si můžete představit jako vazby pro prohledávání indexů ke zjištění oprávněných úvodních klíčů.

Následující informace v příkladu znázorňuje spojení typu self-join:

```
(informix.a.c1 = informix.a.c1 AND informix.a.c2 = informix.a.c2 )
```

Tato informace reprezentuje prohledávání vnitřních indexů. Pro sloupce s hlavními klíči c1 a c2 se použije predikát spojení typu self-join označující, že hodnota sloupce c1 a c2 přichází z prohledávání vnějších indexů. Predikát u sloupce c3 slouží jako filtr indexů, což způsobuje, že je prohledávání vnitřních indexů efektivní.

Běžná prohledávání indexů nepoužívají pro umístění prohledávání indexů filtry u sloupce c3, protože sloupce s hlavním klíčem c1 a c2 nemají predikáty rovnosti.

Direktiva INDEX_SJ vynutí cestu k indexu spojení typu self-join pomocí určeného indexu nebo zvolením indexu ze seznamu indexů s nejnižší nákladovostí, a to i v případě, že pro sloupce s hlavními klíči indexu nejsou k dispozici statistické údaje o distribuci dat. Direktiva AVOID_INDEX_SJ zabrání cestě k indexu spojení typu self-join pro určený index nebo indexy. Další informace naleznete v části “Direktivy přístupové metody” na stránce 11-4 a v příručce *IBM Informix Guide to SQL: Syntax*.

Vyhodnocení plánu dotazů

Optimalizátor zvažuje všechny plány dotazů prostřednictvím analýzy takových faktorů, jako jsou vstup - výstup na disk a nákladovost CPU. Konstruuje všechny vhodné plány souběžně pomocí vyhledávací strategie typu bottom-up a breadth-first. To znamená, že optimalizátor nejprve konstruuje všechny možné páry spojení. Tak je vyloučen nákladnější plán z každého *redundantního* páru, což jsou páry spojení, které obsahují stejné tabulky a vytvářejí stejnou sadu řádků jako druhý pár spojení. Pokud například ani jedno spojení neurčuje uspořádanou sadu řádků pomocí klauzulí ORDER BY či GROUP BY nebo příkazem SELECT, bude pár spojení (A x B) redundantní vzhledem k (B x A).

Pokud dotaz používá dodatečné tabulky, optimalizátor spojí každý zbývající pár s novou tabulkou, aby vytvořil možné triplety spojení a vyloučil tak nejnákladnější z redundantních tripletů. Tento postup se použije pro každou následující tabulku, která má být přidána. Když

byla vygenerována neredundantní sada možných kombinací spojení, optimalizátor vybere plán, který se jeví jako plán s nejnižší nákladovostí provedení.

Sestava, která zobrazuje plán dotazů zvolený optimalizátorem

Každý uživatel, který spustí dotaz, může pomocí příkazu SET EXPLAIN nebo direktivy EXPLAIN zobrazit plán dotazů, který si optimalizátor zvolil. Další informace o tom, jak určit direktivy, naleznete v části “Direktivy EXPLAIN” na stránce 11-10. Uživatel zadá příkaz SET EXPLAIN ON nebo příkaz SET EXPLAIN ON AVOID_EXECUTE před příkazem SQL pro daný dotaz tak, jak je uvedeno v následujícím příkladu.

```
SET EXPLAIN ON AVOID_EXECUTE;
SELECT * FROM customer, orders
WHERE customer.customer_num = orders.customer_num
   AND customer.lname = "Higgins";
```

Pokud uživatel nemá přístup ke zdrojovému kódu jazyka SQL, může administrátor databáze dynamicky nastavit příkaz SET EXPLAIN pomocí příkazu **onmode -Y** spouštějícího kód jazyka SQL. Další informace o příkazu SET EXPLAIN naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Poté, co databázový server vykoná příkaz SET EXPLAIN ON nebo dynamicky nastaví příkaz SET EXPLAIN pomocí příkazu **onmode -Y**, napíše server do souboru pro následující dotazy, které uživatel zadá, vysvětlení týkající se každého plánu dotazů.

Další informace naleznete v části “Soubor sqexplain.out”. Informace o části Query Statistics v souboru **sqexplain.out**, která je užitečným zdrojem pro řešení problémů týkajících se ladění výkonu, naleznete v části “Část Query Statistics poskytuje informace pro ladění výkonu” na stránce 10-11. Úplný popis výstupního souboru uvádí příručka *IBM Informix Guide to SQL: Syntax*.

Soubor sqexplain.out

Jen pro UNIX

V operačním systému UNIX zapíše databázový server výstup příkazu SET EXPLAIN ON nebo direktivy EXPLAIN do souboru **sqexplain.out**.

Pokud je klientská aplikace a databázový server ve stejném počítači, bude soubor **sqexplain.out** uložen do aktuálního adresáře. Jestliže používáte verzi 5.x nebo dřívější klientské aplikace a soubor **sqexplain.out** se nezobrazil v aktuálním adresáři, zkontrolujte, zda se tento soubor nenachází v domovském adresáři.

Když je aktuální databáze v jiném počítači, bude soubor **sqexplain.out** uložen v domovském adresáři na vzdáleném hostiteli.

Konec Jen pro UNIX

Jen pro Windows

V operačním systému Windows zapíše databázový server výstup příkazu SET EXPLAIN ON nebo direktivy EXPLAIN do souboru **%INFORMIXDIR%\sqexpln\username.out**.

Konec Jen pro Windows

Jestliže ke spuštění příkazu SET EXPLAIN používáte příkaz **onmode -Y**, zobrazí se výstup v souboru **sqexplain.out**.**sessionid**. Pokud již soubor sqexplain.out existuje, zastaví databázový server jeho používání, dokud administrátor nevypne dynamický příkaz SET EXPLAIN pro tuto relaci.

Soubor **sqexplain.out** informuje, zda jsou platné externí direktivy.

Část Query Statistics souboru **sqexplain.out** je užitečným zdrojem pro řešení problémů s laděním výkonu. Další informace naleznete v části "Část Query Statistics poskytuje informace pro ladění výkonu".

Úplný popis výstupu uvádí příručka *IBM Informix Guide to SQL: Syntax*.

Část Query Statistics poskytuje informace pro ladění výkonu

Pokud je konfigurační parametr EXPLAIN_STAT povolen, bude v souboru **sqexplain.out** uvedena část Query Statistics, kterou zobrazují příkaz SET EXPLAIN jazyka SQL a příkaz **onmode -Y** *id_relace*.

Část Query Statistics souboru **sqexplain.out** zobrazuje odhadovaný počet řádků, které plán dotazů očekává, že budou vráceny, skutečný počet vrácených řádků a další informace o dotazu. Tyto informace, které znázorňují celkový tok plánu dotazů a počet řádků procházejících každým stupněm dotazu, můžete využít k ladění problémů týkajících se výkonu.

Následující příklad zobrazuje část se statistickými údaji dotazu (část Query statistics) ve výstupu příkazu SET EXPLAIN. Pokud se odhadovaný a skutečný počet řádků prohledávaných nebo spojených zcela liší, mohou být statistické údaje v této tabulce staré a měly by být aktualizovány.

```
select * from tab1, tab2 where tab1.c1 = tab2.c1 and tab1.c3 between 0 and 15

Estimated Cost: 104
Estimated # of Rows Returned: 69

  1) zelaine.tab2: SEQUENTIAL SCAN

  2) zelaine.tab1: INDEX PATH

    (1) Index Keys: c1 c3   (Serial, fragments: ALL)
        Lower Index Filter: (zelaine.tab1.c1 = zelaine.tab2.c1
                             AND zelaine.tab1.c3 >= 0 )
        Upper Index Filter: zelaine.tab1.c3 <= 15
NESTED LOOP JOIN

Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                tab2
  t2                tab1

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     50         50        50         00:00:00   4

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     67         69        4          00:00:00   2

  type     rows_prod  est_rows  time       est_cost
  ------------------------------------------------
  nljoin   67         70        00:00:00   104
```

Obrázek 10-8. Část Query Statistics ve výstupu příkazu SET EXPLAIN.

Další vzorové sestavy naleznete v části “Vzorové sestavy plánu dotazů”.

Vzorové sestavy plánu dotazů

Následující části popisují vzorové plány dotazů, které si můžete zobrazit při analýze výkonu dotazů.

Dotaz do jedné tabulky

Obrázek 10-9 znázorňuje výstup příkazu SET EXPLAIN pro jednoduchý dotaz.

```
QUERY:
------
SELECT fname, lname, company FROM customer

Estimated Cost: 2
Estimated # of Rows Returned: 28

  1) virginia.customer: SEQUENTIAL SCAN
```

Obrázek 10-9. Část výstupu příkazu SET EXPLAIN pro jednoduchý dotaz.

Obrázek 10-10 zobrazuje výstup příkazu SET EXPLAIN pro složitý dotaz do tabulky **customer**.

```
QUERY:
------
SELECT fname, lname, company FROM customer
WHERE company MATCHES 'Sport*' AND
    customer_num BETWEEN 110 AND 115
ORDER BY lname

Estimated Cost: 1
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

  1) virginia.customer: INDEX PATH

        Filters: virginia.customer.company MATCHES 'Sport*'

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: virginia.customer.customer_num >= 110
        Upper Index Filter: virginia.customer.customer_num <= 115
```

Obrázek 10-10. Část výstupu příkazu SET EXPLAIN pro složitý dotaz.

Následující řádky výstupu, které znázorňuje Obrázek 10-10, ukazují rozsah prohledávání indexu pro druhý dotaz:

- Lower Index Filter: virginia.customer.customer_num >= 110

  Spustí prohledávání indexu od hodnoty klíče indexu 110.
- Upper Index Filter: virginia.customer.customer_num <= 115

  Zastaví prohledávání indexu na hodnotě klíče indexu 115.

Dotaz do více tabulek

Obrázek 10-11 znázorňuje výstup příkazu SET EXPLAIN pro dotaz do více tabulek.

```
QUERY:
------
SELECT C.customer_num, O.order_num, SUM (I.total_price)
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
GROUP BY C.customer_num, O.order_num

Estimated Cost: 78
Estimated # of Rows Returned: 1
Temporary Files Required For: Group By

  1) virginia.o: SEQUENTIAL SCAN

  2) virginia.c: INDEX PATH

    (1) Index Keys: customer_num   (Key-Only)  (Serial, fragments: ALL)
        Lower Index Filter:
          virginia.c.customer_num = virginia.o.customer_num
NESTED LOOP JOIN

  3) virginia.i: INDEX PATH

    (1) Index Keys: order_num   (Serial, fragments: ALL)
        Lower Index Filter: virginia.o.order_num = virginia.i.order_num
NESTED LOOP JOIN
```

Obrázek 10-11. Část výstupu příkazu SET EXPLAIN pro dotaz do více tabulek.

Výstup příkazu SET EXPLAIN uvádí přístupový plán a pořadí, ve kterém databázový server přistupuje k tabulkám za účelem čtení tabulky. V plánu, který znázorňuje Obrázek 10-11, je uvedeno, že databázový server má provést následující akce:

1. Databázový server má nejdříve číst tabulku **orders**.

   Jelikož u tabulky **orders** neexistují žádné filtry, musí databázový server přečíst všechny řádky. Nejméně nákladnou metodou je čtení tabulky ve fyzickém pořadí.

2. Pro každý řádek tabulky **orders** má databázový server hledat shodu v tabulce **customer**.

   Vyhledávání používá index u sloupce **customer_num**. Notace Key-Only (pouze klíče) znamená, že se v tabulce **customer** má číst pouze index, protože se pro spojení a výstup použije pouze sloupec **c.customer_num** a tento sloupec je klíč indexu.

3. Pro každý řádek z tabulky **orders**, který má shodný sloupec **customer_num**, má databázový server hledat shodu v tabulce **items** pomocí indexu sloupce **order_num**.

Prohledávání hlavního klíče

Prohledávání hlavního klíče je prohledávání indexu, které používá jiné klíče než ty, které jsou uvedené jako nižší nebo vyšší filtry indexu. Obrázek 10-12 znázorňuje vzorový dotaz používající prohledávání indexu.

```
create index idx1 on tab1(c1, c2);

select * from tab1 where (c1 > 0) and ( (c2 = 1) or (c2 = 2))
Estimated Cost: 4
Estimated # of Rows Returned: 1

1) pubs.tab1: INDEX PATH

    (1) Index Keys: c1 c2 (Key-First) (Serial, fragments: ALL)
   Lower Index Filter: pubs.tab1.c1 > 0
   Index Key Filters:  (pubs.tab1.c2 = 1 OR pubs.tab1.c2 = 2)
```

Obrázek 10-12. Část výstupu příkazu SET EXPLAIN pro prohledávání hlavního klíče.

Ačkoli v tomto příkladu musí databázový server nakonec číst data řádků, aby mohl vrátit výsledky dotazu, pokusí se snížit počet možných řádků tak, že nejdříve použije dodatečné filtry klíče. Databázový server před čtením dat řádku použije pomocí indexu dodatečný filtr, c2 = 1 OR c2 = 2.

Plány dotazů pro poddotazy

Optimalizátor může změnit poddotaz tak, aby se automaticky spojil, pokud toto spojení bude vykazovat nižší nákladovost. Například ve vzorovém výstupu příkazu SET EXPLAIN ON, který znázorňuje Obrázek 10-13, je uvedeno, jak optimalizátor změní tabulku v poddotazu a učiní z ní vnitřní tabulku ve spojení.

```
QUERY:
------
SELECT company, fname, lname, phone
FROM customer c
WHERE EXISTS(
   SELECT customer_num FROM cust_calls u
      WHERE c.customer_num = u.customer_num)

Estimated Cost: 6
Estimated # of Rows Returned: 7

  1) virginia.c: SEQUENTIAL SCAN

  2) virginia.u: INDEX PATH  (First Row)

    (1) Index Keys: customer_num call_dtime   (Key-Only)
                                              (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num
NESTED LOOP JOIN  (Semi Join)
```

Obrázek 10-13. Část výstupu příkazu SET EXPLAIN pro vyrovnaný poddotaz.

Další informace o příkazu SET EXPLAIN ON naleznete v části “Sestava, která zobrazuje plán dotazů zvolený optimalizátorem” na stránce 10-10.

Když optimalizátor změní poddotaz na spojení, může použít několik variant přístupového plánu a plánu spojení:

- Prohledávání prvního řádku

  Prohledávání prvního řádku je variantou prohledávání tabulky. Když databázový server najde shodu, prohledávání tabulky se zastaví.
- Prohledávání s přeskočením duplicitního indexu

  Prohledávání indexu s přeskočením duplicitního indexu je variantou prohledávání indexu. Databázový server neprohledává duplikáty.
- Smíšené spojení

  Smíšené spojení je variantou spojení typu vnořená smyčka. Databázový server zastaví prohledávání vnitřní tabulky po nalezení první shody. Další informace o smíšeném spojení naleznete v části “Spojení typu vnořená smyčka” na stránce 10-3.

Plány dotazu pro tabulky odvozené od kolekce

Tabulka odvozená od kolekce je speciální metoda, kterou databázový server použije ke zpracování dotazu do kolekce. Aby bylo možné používat tabulku odvozenou od kolekce, musí dotaz obsahovat klíčové slovo TABLE v klauzuli FROM příkazu jazyka SQL. Další informace o použití tabulek odvozených od kolekce v příkazu jazyka SQL naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Ačkoli ve skutečnosti databáze nevytváří pro kolekci tabulku, zpracovává data tak, jako by se jednalo o tabulku. Tabulky odvozené od kolekce v některých případech umožňují vývojářům použít při přístupu do kolekce méně kurzorů a hostitelských proměnných.

Tyto příkazy jazyka SQL vytvářejí sloupec kolekce označený jako podřízený, neboli **children**:

```
CREATE ROW TYPE person(name CHAR(255), id INT);
CREATE TABLE parents(name CHAR(255),
id INT,
children LIST(person NOT NULL));
```

Následující dotaz vytvoří tabulku odvozenou od kolekce pro podřízený sloupec, označený **children**, a bude s prvky této kolekce zacházet jako s řádky tabulky:

```
SELECT name, id
FROM TABLE(MUTLISET(SELECT children
FROM parents
WHERE parents.id
= 1001)) c_table(name, id);
```

Alternativně můžete zadat tabulku odvozenou od kolekce v klauzuli FROM, jak uvádí tento příklad:

```
SELECT name, id
FROM (SELECT children
FROM parents
WHERE parents.id
= 1001) c_table(name, id);
```

Příklad znázorňující, jak databázový server dokončí dotaz: Při dokončování dotazu provádí databázový server kroky uvedené v tomto příkladu:

1. Prohledá nadřazenou tabulku, označenou **parents**, aby nalezl řádek, ve kterém je parents.id = 1001

   Tato operace je uvedena jako SEQUENTIAL SCAN ve výstupu příkazu SET EXPLAIN, který znázorňuje Obrázek 10-14.
2. Čte hodnotu sloupce kolekce nazvaného **children**.
3. Prohledá jednu kolekci a aplikaci vrátí hodnotu **name** a **id**.

   Tato operace je ve výstupu příkazu SET EXPLAIN uvedena jako COLLECTION SCAN, jak znázorňuje Obrázek 10-14.

```
QUERY:
------
SELECT name, id
FROM (SELECT children
FROM parents
WHERE parents.id
= 1001) c_table(name, id);

Estimated Cost: 2
Estimated # of Rows Returned: 1

  1) lsuto.c_table: COLLECTION SCAN
    Subquery:
    ---------
    Estimated Cost: 1
    Estimated # of Rows Returned: 1

      1) lsuto.parents: SEQUENTIAL SCAN

            Filters: lsuto.parents.id = 1001
```

Obrázek 10-14. Plán dotazů používající tabulku odvozenou do kolekce.

Odvozené tabulky složené do nadřazených dotazů: Výkon odvozených tabulek můžete zvýšit použitím jazyka SQL při složení odvozených tabulek v jednoduchých dotazech do nadřazeného dotazu místo do výsledků dotazu, které se umisťují do dočasné tabulky. Jazyk SQL použijte stejným způsobem jako v tomto příkladu:

```
select * from ((select col1, col2 from tab1)) as vtab(c1,c2)
```

Pokud je ovšem dotaz složitý (zahrnuje souhrny, operace ORDER BY nebo operaci UNION), vytvoří server dočasnou tabulku.

Databázový server složí odvozené tabulky způsobem, který se podobá způsobu, jakým server skládá pohledy prostřednictvím konfiguračního parametru (jak je popsáno v části "Povolení skládání pohledů za účelem zvýšení výkonu dotazů" na stránce 13-19). Pokud je konfigurační parametr IFX_FOLDVIEW povolen, pohledy jsou skládány do nadřazeného dotazu. Pohledy nejsou skládány do výsledků dotazu umístěných do dočasné tabulky.

V následujícím příkladu jsou uvedeny odvozené tabulky složené do hlavního dotazu.

```
select * from ((select vcol0, tab1.col1 from
        table(multiset(select col2 from tab2 where col2 > 50 ))
 vtab2(vcol0),tab1 )) vtab1(vcol1,vcol2)
 where vcol1 = vcol2

Estimated Cost: 2
Estimated # of Rows Returned: 1

  1) informix.tab2: SEQUENTIAL SCAN

        Filters: informix.tab2.col2 > 50

  2) informix.tab1: SEQUENTIAL SCAN

        Filters:
        Table Scan Filters: informix.tab1.col1 > 50

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.tab2.col2 = informix.tab1.col1
```

Obrázek 10-15. Plán dotazů používající odvozené tabulky složené do nadřazeného dotazu.

```
select * from (select col1 from tab1 where col1 = 100) as vtab1(c1)
left join (select col1 from tab2 where col1 = 10) as vtab2(vc1)
on  vtab1.c1 =  vtab2.vc1

Estimated Cost: 5
Estimated # of Rows Returned: 1

      1) informix.tab1: SEQUENTIAL SCAN

            Filters: informix.tab1.col1 = 100

      2) informix.tab2: AUTOINDEX PATH

        (1) Index Keys: col1   (Key-Only)
            Lower Index Filter: informix.tab1.col1 = informix.tab2.col1
            Index Key Filters:  (informix.tab2.col1 = 10 )

    ON-Filters:(informix.tab1.col1 = informix.tab2.col1
            AND informix.tab2.col1 = 10 )
    NESTED LOOP JOIN(LEFT OUTER JOIN)
```

Obrázek 10-16. Druhý plán dotazů používající odvozené tabulky složené do nadřazeného dotazu.

Následující příkaz zobrazuje složitý dotaz zahrnující operaci UNION. V tomto případě byla vytvořena dočasná tabulka.

```
select * from (select col1 from tab1 union select col2 from tab2 )
as vtab(vcol1)  where vcol1 < 50

Estimated Cost: 4
Estimated # of Rows Returned: 1

  1) (Temp Table For Collection Subquery): SEQUENTIAL SCAN
```

Obrázek 10-17. Složitý dotaz používající odvozené tabulky, který vytvoří dočasnou tabulku.

Faktory, které ovlivňují plán dotazů

Když optimalizátor určuje plán dotazů, přiřadí jednotlivým možným plánům nákladovost a potom zvolí plán s nejnižší nákladovostí. Mezi faktory, které optimalizátor používá k určení nákladovosti každého plánu dotazů, patří tyto:

- Počet požadavků na vstup - výstup, které jsou přidruženy ke každému přístupu k systému souborů.
- Práce CPU vyžadovaná k určení, které řádky odpovídají predikátu dotazu.
- Zdroje požadované k řazení nebo seskupení dat.
- Množství paměti dostupné pro dotaz (určené parametry DS_TOTAL_MEMORY a DS_MAX_QUERIES).

Při výpočtu nákladovosti každého možného plánu dotazu postupuje optimalizátor následujícím způsobem:

- Použije sadu statistických údajů, která popisuje charakter a fyzické charakteristiky tabulkových dat a indexů.
- Prověří filtry dotazu.
- Prověří indexy, které by mohly být použity v plánu.
- Použije nákladovost přesunu dat k provedení lokálního nebo vzdáleného spojení pro distribuované dotazy.

Statistické údaje uchovávané pro tabulku a index

Přesnost, s jakou může optimalizátor dotazů posoudit nákladovost provedení u plánu dotazů, závisí na informacích, které má k dispozici. Pomocí příkazu UPDATE STATISTICS můžete udržovat jednoduché statistické údaje o tabulce a k ní přidružených indexech. Aktualizované statistické údaje poskytují optimalizátoru dotazů informace, které mohou minimalizovat množství času potřebného k provedení dotazů na tuto tabulku.

Databázový server spustí statistický profil tabulky v okamžiku vytváření tabulky a tento profil je aktualizován po vydání příkazu UPDATE STATISTICS. Optimalizátor dotazů automaticky nepřepočítává profil pro tabulky. V některých případech může shromažďování statistických údajů trvat déle než provedení dotazu.

Chcete-li zajistit, aby optimalizátor vybral plán dotazů, který nejlépe odráží aktuální stav tabulek, spouštějte v pravidelných intervalech příkaz UPDATE STATISTICS. Pokyny naleznete v části “Aktualizace statických údajů, nejsou-li generovány automaticky” na stránce 13-7.

Optimalizátor použije následující informace systémového katalogu a vytvoří plán dotazů:

- Počet řádků v tabulce stanovený nejaktuálnějším příkazem UPDATE STATISTICS.
- Zda je sloupec vymezený tak, aby byl jedinečný.

- Distribuce hodnot sloupce, pokud je požadována klíčovým slovem MEDIUM nebo HIGH v příkazu UPDATE STATISTICS.

  Další informace o distribucích dat naleznete v části “Vytvoření distribucí dat” na stránce 13-9.
- Počet stránek na disku, které obsahují data řádku.

Optimalizátor také použije následující informace systémového katalogu a vytvoří plán dotazů:

- Indexy existující v tabulce (včetně sloupců, které indexují) a skutečnost, zda jsou řazeny vzestupně či sestupně a zda jsou v klastru.
- Hloubka struktury indexů (měřítko množství práce, kterou je nutné provést při vyhledávání indexu).
- Počet stránek na disku, které jsou obsazeny položkami indexu.
- Počet jedinečných položek v indexu, který je možno použít k odhadu řádků vracených filtrem rovnosti.
- Druhá největší a druhá nejmenší hodnota klíče v indexovaném sloupci.

Zvažovány jsou pouze druhá největší a druhá nejmenší hodnota klíče, protože extrémní hodnoty mohou mít zvláštní význam, který se nevztahuje ke zbytku dat ve sloupci. Databázový server předpokládá, že hodnoty klíče jsou distribuovány mezi druhou největší a druhou nejmenší hodnotou. Ukládají se pouze úvodní 4 bajty těchto klíčů. Pokud vytvoříte distribuci pro sloupec přidružený k indexu, použije optimalizátor tuto distribuci při odhadu počtu řádků, které vyhovují dotazu.

Více informací o katalozích systémových tabulek naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Filtry dotazu

Optimalizátor dotazů zakládá odhady nákladovosti dotazů na počtu řádků, které mají být vyhledány v každé tabulce. Odhadovaný počet řádků je naopak založen na *selektivitě* každého výrazu podmínky, který je použit v rámci klauzule WHERE. Výraz podmínky, který se použije k výběru řádků, je označován jako *filtr*.

Selektivita je hodnota mezi 0 a 1, která označuje poměr řádků v tabulce, které mohou projít filtrem. Selektivní filtr, přes který projde málo řádků, má selektivitu blízkou hodnotě 0 a filtr, který propustí téměř všechny řádky, má selektivitu blízkou hodnotě 1. Pokyny týkající se filtrů naleznete v části “Zlepšení selektivity filtrů” na stránce 13-3.

Optimalizátor může pomocí distribucí dat vypočítat selektivitu filtrů v dotazu. Ovšem v případě, kdy distribuce dat neexistují, vypočítá databázový server selektivitu pro filtry různých typů na základě indexů tabulky. Následující tabulka uvádí některé z hodnot selektivity, které optimalizátor přiřadil filtrům různého typu. Hodnoty selektivity vypočítané pomocí distribucí dat jsou dokonce přesnější než hodnoty uváděné v této tabulce.

Výraz filtru	Selektivita (F)
indexovaný-sl = *hodnota-literáluindexovaný-sl* = *hostitelská-proměnnáindexovaný-sl* IS NULL	F = 1/(počet různých klíčů v indexu)
tab1.indexovaný-sl = *tab2.indexovaný-sl*	F = 1/(počet různých klíčů ve větším indexu)
indexovaný-sl > *hodnota-literálu*	F = (druhý-max - *hodnota_literálu*)/(druhý-max - druhý-min)
indexovaný-sl < *hodnota-literálu*	F = (*hodnota_literálu* - druhý-min)/(druhý-max - druhý-min)
libovolný-sl IS NULL*libovolný-sl* = *libovolný-výraz*	F = 1/10
libovolný-sl > *libovolný-výrazlibovolný-sl* < *libovolný-výraz*	F = 1/3
libovolný-sl MATCHES *libovolný-výrazlibovolný-sl* LIKE *libovolný-výraz*	F = 1/5
EXISTS *poddotaz*	F = 1 if *poddotaz* estimated to return >0 rows, else 0
NOT *výraz*	F = 1 - F(*výraz*)
výraz1 AND *výraz2*	F = F(*výraz1*) x F(*výraz2*)
výraz1 OR *výraz2*	F = F(*výraz1*) + F(*výraz2*) - (F(*výraz1*) x F(*výraz2*))
libovolný-sl IN *seznam*	Treated as *libovolný-sl* = *položka*$_1$ OR . . . OR *libovolný-sl* = *položka*$_n$.
libovolný-sl relop ANY *poddotaz*	Treated as hodnota relop libovolného sloupce$_1$ OR . . . OR *hodnota-relop-libovolného sloupce*$_n$ pro odhadovanou velikost poddotazu *n*. Hodnota *relop* zde představuje libovolný relační operátor, jako například <, >, >=, <=.

Klíč:

- *indexovaný-sl*: první nebo jediný sloupec v indexu.
- *druhý-max*, *druhý-min*: druhá největší a druhá nejmenší hodnota klíče ve sloupci s indexem.

V této tabulce představuje znaménko plus (+) logické sjednocení (= logický operátor OR) a symbol násobení (x) znamená logický součin (= logický operátor AND).

Indexy pro vyhodnocení filtru

Optimalizátor dotazů bere v úvahu, zda je možné index použít k vyhodnocení filtru. Pro tento účel je indexovaný sloupec jediný sloupec s indexem nebo první sloupec jmenovaný v složeném indexu. Pokud hodnoty obsažené v indexu jsou vše, co je požadováno, řádky nejsou čteny. U datových stránek je rychlejší vynechat vyhledávání stránek vždy, když může databázový server číst hodnoty přímo z indexu.

Optimalizátor může zvolit index v libovolném z následujících případů:

- Je-li sloupec indexovaný a hodnota, která má být porovnávána, je literál, hostitelská proměnná nebo nesouvztažný poddotaz.

  Databázový server může vyhledat odpovídající řádky v tabulce tak, že nejprve nalezne řádek v příslušném indexu. Pokud není příslušný index k dispozici, musí databázový server prohledat každou tabulku v celém rozsahu.
- Je-li sloupec indexovaný a hodnota, která má být porovnávána, je sloupec v jiné tabulce (výraz spojení).

  Databázový server může použít index k nalezení shodných hodnot. Příkladem může být následující výraz spojení:

  ```
  WHERE customer.customer_num = orders.customer_num
  ```

  Pokud jsou řádky tabulky **customer** čteny nejdříve, hodnoty **customer_num** mohou být použity u indexu ve sloupci **orders.customer_num**.

- Při zpracování klauzule ORDER BY.

  Pokud se všechny sloupce v klauzuli zobrazují v požadované posloupnosti v rámci jednoho indexu, může databázový server použít tento index ke čtení řádků v jejich uspořádané posloupnosti, a tím se vyhnout řazení.
- Při zpracování klauzule GROUP BY.

  Pokud se všechny sloupce v klauzuli zobrazují v jednom indexu, může databázový server číst skupiny se stejnými klíči z indexu, aniž by požadoval další zpracování poté, co jsou řádky získány z jejich tabulek.

Vliv funkce PDQ na plán dotazů

Když je funkce PDQ (Parallel database query) zapnuta, může optimalizátor zvolit současné provádění dotazu, což může výrazným způsobem zvýšit výkon, pokud databázový server zpracovává dotazy, které jsou vyvolány aplikacemi podporujícími rozhodování. Další informace popisuje Kapitola 12, “Paralelní databázový dotaz”, na stránce 12-1.

Vliv nastavení hodnoty OPTCOMPIND na plán dotazů

Nastavení OPTCOMPIND ovlivňuje *přístupový plán*, který si optimalizátor volí pro dotazy do jedné nebo více tabulek tak, jak je popsáno v následujících částech.

Hodnotu OPTCOMPIND v rámci relace můžete změnit pomocí příkazu SET ENVIRONMENT OPTCOMPIND. Další informace o použití tohoto příkazu naleznete v části “Použití příkazu ENVIRONMENT OPTCOMPIND pro nastavení hodnoty proměnné OPTCOMPIND v rámci relace” na stránce 3-10.

Dotaz do jedné tabulky

V případě prohledávání jedné tabulky, když je hodnota OPTCOMPIND nastavena na 0 nebo 1 a aktuální úroveň odstínění (izolace) transakcí je opakovatelné čtení, zvažuje optimalizátor následující přístupové plány:

- Pokud je k dispozici index, použije jej optimalizátor k přístupu k tabulce.
- Pokud není žádný index k dispozici, zváží optimalizátor prohledávání tabulky ve fyzickém pořadí.

Pokud při konfiguraci databázového serveru není hodnota OPTCOMPIND nastavena, má výchozí hodnotu 2. Jestliže je hodnota OPTCOMPIND nastavena na hodnotu 2 nebo na 1 a aktuální úroveň izolace není opakovatelné čtení, zvolí optimalizátor nejméně nákladný plán k přístupu k tabulce.

Dotaz do více tabulek

U plánů spojení ovlivňuje nastavení hodnoty OPTCOMPIND přístupový plán pro specifické uspořádané páry tabulek. Nastavte hodnotu OPTCOMPIND na 0, pokud chcete, aby databázový server vybral metodu spojení přesně stejným způsobem jako v předchozích verzích databázového serveru. Tato volba zajišťuje kompatibilitu s předchozími verzemi.

Je-li hodnota OPTCOMPIND nastavena na 0 nebo na 1 a aktuální úroveň izolace je opakovatelné čtení, dá optimalizátor přednost spojení typu vnořená smyčka.

Důležité: Je-li hodnota OPTCOMPIND nastavena na 0, optimalizátor nezvolí spojení typu hash.

Pokud je hodnota OPTCOMPIND nastavena na 2 nebo na 1 a úroveň odstínění (izolace) transakcí není opakovatelné čtení, zvolí optimalizátor nejméně nákladný plán dotazů mezi dříve uvedenými plány, přičemž nebude upřednostňovat spojení typu vnořená smyčka.

Vliv dostupné paměti na plán dotazů

Databázový server omezuje množství paměti, které může paralelní dotaz používat, na základě hodnot parametrů DS_TOTAL_MEMORY a DS_MAX_QUERIES. Pokud je množství dostupné paměti pro dotaz příliš nízké, aby bylo možné vykonat spojení typu hash, databázový server použije místo toho spojení typu vnořená smyčka.

Další informace o paralelních dotazech a parametrech DS_TOTAL_MEMORY a DS_MAX_QUERIES popisuje Kapitola 12, “Paralelní databázový dotaz”, na stránce 12-1.

Časová nákladovost dotazu

Tato část vysvětluje vliv doby odezvy akcí, které provádí databázový server při zpracování dotazu.

Mnohé z nákladovostí, které jsou popsány, nelze snížit úpravou konstrukce dotazu. Některé však sníženy být mohou. Optimální konstrukcí dotazů a vhodnými indexy lze snížit následující nákladovosti:

- Čas řazení
- Nevhodná spojení dat
- Změny na místě pomocí příkazu ALTER TABLE
- Vyhledávání indexů

Další informace o tom, jak optimalizovat specifické dotazy, popisuje Kapitola 13, “Zvyšování výkonu jednotlivých dotazů”, na stránce 13-1.

Nákladovost aktivity paměti

Databázový server může zpracovat pouze data v paměti. Musí načíst řádky do paměti, aby tyto řádky vyhodnotil vůči filtrům dotazu. Jakmile databázový server zjistí, že řádky těmto filtrům vyhovují, připraví výstupní řádek v paměti tak, že shromáždí vybrané sloupce.

Většina těchto aktivit je prováděna rychle. V závislosti na počítači a jeho pracovní zátěži může databázový server každou sekundu provádět stovky či dokonce tisíce porovnání. V důsledku to znamená, že čas strávený prací v paměti představuje obvykle malou část doby provedení dotazu.

Ačkoli některé aktivity v paměti (například řazení) zabírají značné množství času, trvá mnohem déle načíst řádek z disku, než prohlédnout každý řádek, který je již v paměti.

Časová nákladovost řazení

Řazení vyžaduje práci v paměti, stejně jako práci s diskem. Práce uvnitř paměti závisí na množství sloupců, které jsou řazeny, šířce kombinovaného řadicího klíče a počtu kombinací řádků, které projdou filtrem dotazu. Pomocí následujícího vzorce můžete vypočítat práci uvnitř paměti, kterou požaduje operace řazení:

$$W_m = (c * N_{fr)} + (w * N_{fr} \log_2(N_{fr}))$$

W_m je práce uvnitř paměti.

c je počet sloupců, které se mají seřadit, a představuje nákladovost vyjmutí hodnot sloupce z řádku a jejich sloučení do klíče řazení.

w je proporcionální se šířkou kombinovaného klíče řazení v bajtech a zastupuje práci při kopírování nebo porovnávání jednoho klíče řazení. Numerická hodnota pro w výrazně závisí na používaném hardwaru počítače.

N_{fr} je počet řádků, které projdou filtrem dotazu.

Pokud je množství dat, které mají být seřazeny, příliš velké, může řazení zahrnovat dočasný zápis informací na disk. Zápisy na disk můžete nasměrovat tak, aby k nim docházelo v prostoru souborů nebo prostoru dbspace, který spravuje databázový server. Podrobnosti naleznete v části “Konfigurace prostorů dbspace pro dočasné tabulky a soubory řazení” na stránce 5-7.

Práce s diskem závisí na počtu stránek na disku, ve kterých se nachází řádky, na počtu řádků, které vyhovují podmínkám predikátu dotazu, na počtu řádků, které mohou být umístěny do uspořádané stránky, a na počtu operací sloučení, které musí být provedeny. Pomocí následujícího vzorce můžete vypočítat práci s diskem, kterou požaduje operace řazení:

$W_d = p + (N_{fr}/N_{rp}) * 2 * (m - 1))$

W_d	je práce s diskem.
p	je počet stránek na disku.
N_{fr}	je počet řádků, které projdou filtry.
N_{rp}	je počet řádků, které mohou být umístěny na stránku.
m	představuje počet *úrovní sloučení*, které musí řazení použít.

Faktor *m* závisí na počtu klíčů řazení, které mohou být uchovány v paměti. Pokud zde nejsou žádné filtry, hodnota N_{fr}/N_{rp} se rovná hodnotě *p*.

Pokud mohou být v paměti uchovány všechny klíče, *m*=1 a práce s diskem se rovná hodnotě *p*. Jinými slovy, řádky jsou čteny a řazeny v paměti.

V případě průměrných až velkých tabulek jsou řádky řazeny v dávkách, které se vejdou do paměti, a potom jsou tyto dávky sloučeny. Pokud se hodnota *m*=2, řádky jsou čteny, řazeny a zapisovány v dávkách. Potom jsou tyto dávky znovu čteny a slučovány, což má za následek práci s diskem odpovídající následující hodnotě:

$W_d = ooop + (2 * (N_{fr}/N_{rp}))$

Čím specifičtější jsou filtry, tím méně řádků je řazeno. Se vzrůstajícím počtem řádků a snižujícím se množstvím paměti se zvyšuje práce s diskem.

Chcete-li snížit nákladovost řazení, použijte následující metody:

- Vytvořte co možná nejvíce specifické (selektivní) filtry.
- Omezte seznam projekce na sloupce, které jsou relevantní pro váš problém.

Nákladovost čtení řádků

Když databázový server potřebuje prohlédnout řádek, který ještě není v paměti, musí tento řádek přečíst na disku. Databázový server nečte pouze jeden řádek, čte celou stránku, která tento řádek obsahuje. Pokud tento řádek přesahuje na další stránky, čte všechny tyto stránky.

Skutečná nákladovost čtení stránky je proměnlivá a lze ji obtížně předvídat. Jedná se o kombinaci následujících faktorů:

Faktor	**Vliv faktoru**
Použití vyrovnávací paměti	Pokud je potřebná stránka již ve vyrovnávací paměti stránky, bude nákladovost čtení blízká nule.
Soupeření	Pokud dvě nebo více aplikací požadují přístup k hardwaru disku, mohou mít požadavky na vstup - výstup prodlevu.
Čas vyhledávání	Nejpomalejší činnost, kterou disk provádí, je *vyhledávání* (seek), tzn.

přesouvání přístupového ramena na stopu, která uchovává data. Čas vyhledávání závisí na rychlosti disku a umístění diskového ramenu v okamžiku zahájení operace. Čas vyhledávání kolísá od nuly do téměř jedné sekundy.

Latence — Přenos nemůže začít, dokud se začátek stránky nenatočí pod přístupové rameno. Tato *latence*, neboli rotační zpoždění, závisí na rychlosti disku a na pozici disku v okamžiku zahájení operace. Latence může kolísat od nuly do několika milisekund.

Časová nákladovost čtení stránky může kolísat od mikrosekund (u stránky, která je již ve vyrovnávací paměti) přes několik milisekund (v případě, že soupeření je nulové a diskové rameno je již ve vhodné pozici) ke stovkám milisekund (při soupeření stránky a je-li diskové rameno nad vzdáleným cylindrem disku).

Nákladovost sekvenčního přístupu

Nákladovost disku je nižší, pokud databázový server čte řádky tabulky ve fyzickém pořadí. Když je požadován první řádek stránky, je stránka disku načtena do stránky vyrovnávací paměti. Jakmile je stránka načtena, není třeba ji číst znovu. Požadavky na následující řádky na této stránce jsou plněny z vyrovnávací paměti, dokud nejsou zpracovány všechny řádky na této stránce. Když je stránka vyčerpána, musí být načtena stránka s další sadou řádků. Chcete-li se ujistit, že je následující stránka již v paměti, použijte konfigurační parametr pro dopředné čtení, který je popsaný v části “Parametry RA_PAGES a RA_THRESHOLD” na stránce 5-25.

Pokud pro prostory dbspace používáte zařízení bez vyrovnávací paměti a tabulka je řádně uspořádaná, budou stránky disku se za sebou následujícími řádky umístěny do za sebou následujících umístění na disku. Toto uspořádání umožňuje, aby se při sekvenčním čtení pohybovalo přístupové rameno velmi málo. Kromě toho je při sekvenčním čtení obvykle nižší i nákladovost latence.

Nákladovost nesekvenčního přístupu

Kdykoli je tabulka čtena v náhodném pořadí, jsou požadovány dodatečné přístupy na disk, aby bylo možné číst řádky v požadovaném pořadí. Nákladovost disku je vyšší, když jsou řádky tabulky čteny v pořadí, která nesouvisí s fyzickým uspořádáním na disku. Jelikož stránky nejsou čteny z disku sekvenčně, dochází před čtením každé stránky k prodlevám způsobeným vyhledáváním i rotací. V důsledku toho je čas přístupu na disk mnohem vyšší v případě, kdy diskové zařízení čte stránky tabulky v nesekvenčním pořadí, než když čte tutéž tabulku postupně.

K nesekvenčními přístup obvykle dochází, když k vyhledání řádků použijete index. Ačkoli položky indexu jsou sekvenční, neexistuje žádná záruka, že řádky se sousedními položkami indexu se musí nacházet na stejných (nebo sousedních) datových stránkách. V mnoha případech musí být prováděny samostatné přístupy na disk, aby byla získána stránka pro každý řádek vyhledávaný pomocí indexu. Pokud je tabulka větší než vyrovnávací paměti stránky, musí být stránka, která obsahovala dříve čtený řádek, vyčištěna (odstraněna z vyrovnávací paměti a zapsána zpět na disk) předtím, než je možné zpracovat následující požadavek na řádek na této stránce. Tuto stránku může být nutné znovu číst.

V závislosti na relativním uspořádání tabulky s ohledem na index, můžete někdy vyhledat stránky, které obsahují několik požadovaných řádků. Stupeň, do jakého fyzické uspořádání řádků na disku koresponduje s pořadím řádků v indexu, se nazývá *klastrování*. Vysoce klastrovaná tabulka je taková tabulka, ve které fyzické uspořádání na disku těsně koresponduje s indexem.

Nákladovost vyhledávání indexu

Databázový server zaznamená další nákladovost při vyhledávání řádku prostřednictvím indexu. Index je uložen na disku a jeho stránky musí být načteny do paměti spolu s datovými stránkami, které obsahují požadované řádky.

Vyhledávání indexu je prováděno od kořenové stránky do listové stránky. Kořenová stránka, která je používaná velmi často, se téměř vždycky nachází ve vyrovnávací paměti stránky. Šance nalézt listovou stránku ve vyrovnávací paměti závisí na velikosti indexu, formátu dotazu a frekvenci duplikace hodnoty sloupce. Pokud se každá hodnota vyskytuje v indexu pouze jednou a dotaz je spojení, vyžaduje každý řádek, který má být spojen, nesekvenční vyhledávání v indexu, následované nesekvenčním přístupem k přidruženému řádku v tabulce.

Čtení duplicitních hodnot z indexu

Použití indexu vyvolává další nákladovost pro duplicitní hodnoty během postupného čtení tabulky. Každá položka nebo sada položek se stejnou hodnotou musí být vyhledána v indexu. Potom musí být pro každou položku v tomto indexu proveden náhodný přístup do tabulky, aby bylo možné číst přidružený řádek. Pokud ovšem bude pro odlišnou hodnotu indexu existovat mnoho duplicitních řádků a přidružená tabulka bude vysoce klastrovaná, bude přidaná nákladovost spojení prostřednictvím indexu malá.

Hledání ve sloupcích typu NCHAR nebo NVARCHAR v indexu

Globální podpora jazyků

Indexy vytvořené na sloupcích typu NCHAR nebo NVARCHAR jsou řazeny pomocí pomocí porovnávací hodnoty závislé na národním prostředí. Například se španělským znakem dvojité l (ll) je možné zacházet jako s jedním jedinečným znakem namísto páru l.

V některých národních prostředích není porovnávací hodnota založena na pořadí kódové sady. Vytváření indexů používá k uložení hodnot klíčů do indexu porovnávací hodnotu závislou na národním prostředí. Důsledkem je, že dotaz používající index u datových typů NCHAR nebo NVARCHAR prohledává celý index, protože databázový server vyhledává index v pořadí kódové sady.

Konec Globální podpora jazyků

Nákladovost změny tabulky na místě příkazem ALTER TABLE

Za určitých podmínek používá databázový server algoritmus změny na místě ke změně každého řádku při provádění příkazu ALTER TABLE (místo během operace změna tabulky). Po operaci změna tabulky vloží databázový server řádky pomocí nejnovější definice.

Pokud dotaz přistupuje k řádkům, které nejsou dosud převedeny na novou definici tabulky, můžete si všimnout mírného poklesu výkonu jednotlivého dotazu, protože databázový server provádí před vrácením každého řádku přeformátování tohoto řádku v paměti.

Další informace o podmínkách a přínosech týkajících se výkonu při provádění změn na místě, naleznete v části “Úprava definice tabulky” na stránce 6-32.

Nákladovost zobrazení

Pohledy na tabulku můžete vytvořit z řady důvodů:

- Omezit data, ke kterým může mít uživatel přístup
- Zkrátit čas nutný pro zápis složitého dotazu
- Skrýt složitost dotazu, který uživatel potřebuje napsat

Dotaz na pohled se ovšem může provádět mnohem pomaleji, než bylo očekáváno, pokud složitost definice pohledu způsobí, že je pro zpracování dotazu vytvářena dočasná tabulka. Tato dočasná tabulka je označována jako *materializovaný pohled*. Můžete například vytvořit pohled se sjednocením, a tak sloučit výsledky z několika příkazů SELECT.

Následující vzorový příkaz jazyka SQL vytvoří pohled, který zahrnuje tato sjednocení:

```
CREATE VIEW view1 (col1, col2, col3, col4)
   AS
      SELECT a, b, c, d
         FROM tab1 WHERE
      UNION
      SELECT a2, b2, c2, d2
         FROM tab2 WHERE º
...
      UNION
      SELECT an, bn, cn, dn
         FROM tabn WHERE º
;
```

Když vytvoříte pohled obsahující složité příkazy SELECT, nemusí se koncový uživatel zabývat touto složitostí. Koncový uživatel může pouze napsat jednoduchý dotaz (podobný dotazu v následujícím příkladu):

```
SELECT a, b, c, d
   FROM view1
      WHERE a < 10;
```

Tento dotaz na pohled **view1** se možná může provést pomaleji, než je očekáváno, protože databázový server vytvoří před provedením dotazu pro tento pohled fragmentovanou dočasnou tabulku.

Jinou situací, ve které se může provést dotaz mnohem pomaleji, než je očekáváno, je použití pohledu se spojením typu ANSI. Složitost definice pohledu může způsobit vytvoření dočasné tabulky.

Chcete-li určit, zda se jedná o dotaz, který musí za účelem zpracování pohledu vytvořit dočasnou tabulku, spusťte příkaz SET EXPLAIN. Pokud je ve výstupním souboru příkazu SET EXPLAIN uvedeno Temp Table For View, dotaz potřebuje ke zpracování pohledu dočasnou tabulku.

Nákladovost malých tabulek

Tabulka je malá, pokud se nachází na tak malém počtu stránek, že může být celá uchována ve vyrovnávacích pamětech stránky. Operace v malých tabulkách jsou obecně rychlejší než operace ve velkých tabulkách.

Příklad: V databázi **stores_demo** má tabulka **state**, která uvádí ve vztah zkratky k názvům států, celkovou velikost menší než 1000 bajtů a vejde se na méně než dvě stránky. Tuto tabulku je možné zahrnout při malé nákladovosti do libovolného dotazu. Bez ohledu na to, jak je tato tabulka použita, její načtení z disku vyžaduje maximálně dva přístupy na disk v prvním okamžiku, kdy je požadována.

Nákladovost nevhodného spojení dat

Příkaz jazyka SQL může zaznamenat dodatečnou nákladovost, pokud se datový typ použitý v podmínce liší od definice sloupce v příkazu CREATE TABLE.

Například následující dotaz obsahuje podmínku, která porovnává sloupec s hodnotou datového typu, která se liší od definice tabulky:

```
CREATE TABLE table1 (a integer, );
SELECT * FROM table1
   WHERE a = '123';
```

Databázový server tento dotaz před provedením přepíše, aby převedl hodnotu 123 na celé číslo (integer). Výstup příkazu SET EXPLAIN zobrazuje dotaz v upraveném formátu. Tento převod dat nezpůsobí znatelnou režii.

Dodatečná nákladovost nevhodného spojení dat dosahuje nejvyšších hodnot, když dotaz porovnává znakový sloupec s hodnotou jinou než znakovou a délka čísla se nerovná délce znakového sloupce. Následující dotaz například obsahuje podmínku klauzule WHERE, která kvůli chybějícím uvozovkám porovnává znakový sloupec s hodnotou celého čísla (integer):

```
CREATE TABLE table2 (char_col char(3), );
SELECT * FROM table2
   WHERE char_col = 1;
```

Tento dotaz nalézá jako platné pro hodnotu **char_col** všechny následující hodnoty:

```
' 1'
'001'
'1'
```

Tyto hodnoty nejsou nezbytně klastrované dohromady v klíčích indexu. Z tohoto důvodu neposkytuje index rychlý a správný způsob k získání dat. Výstup příkazu SET EXPLAIN zobrazuje sekvenční prohledávání této situace.

Upozornění: Databázový server nepoužívá index v případě, kdy příkaz jazyka SQL porovnává znakový sloupec s jinou než znakovou hodnotou, jejíž délka se nerovná délce sloupce.

Nákladovost šifrovaných hodnot

Šifrovaná hodnota používá více paměťového prostoru než hodnota prostého textu, neboť spolu se šifrovanou hodnotu jsou uloženy všechny informace potřebné k jejímu dešifrování vyjma šifrovacího klíče. Většina šifrovaných dat požaduje přibližně o 33 procent větší paměťový prostor než nešifrovaná data. Vynecháním pokynu použitého s heslem můžete snížit režii šifrování až o 50 bajtů. Pokud používáte šifrované hodnoty, musíte se ujistit, že máte pro tyto hodnoty dostatek místa.

Další informace o šifrování naleznete v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Nákladovost funkčnosti GLS

Jak je popsáno v části “Hledání ve sloupcích typu NCHAR nebo NVARCHAR v indexu” na stránce 10-25, indexování určitých datových sad způsobí výrazný pokles výkonu. Řazení určitých datových sad také může způsobit pokles výkonu.

Pokud nepotřebujete porovnávací posloupnost jiných znaků než ASCII, doporučujeme, abyste pro znakové sloupce používali datové typy CHAR a VARCHAR všude, kde je to možné. Jelikož data typu CHAR a VARCHAR vyžadují jednoduché porovnávání hodnot, je řazení a indexování těchto sloupců mnohem méně nákladné než u datových typů jiných než ASCII (například typů NCHAR nebo NVARCHAR). Další informace o dalších znakových datových typech naleznete v příručce *IBM Informix GLS User's Guide*.

Nákladovost přístupu k síti

Přesun dat prostřednictvím sítě vyvolá další prodlevy navíc kromě prodlev, které se vyskytují při přímém přístupu na disk. K prodlevám sítě může docházet, když aplikace odešle dotaz

nebo požadavek na aktualizaci prostřednictvím sítě k databázovému serveru v jiném počítači. Ačkoli databázový server provede dotaz v počítači vzdáleného hostitele, tento databázový server vrátí výstup aplikaci prostřednictvím sítě.

Data odeslaná prostřednictvím sítě se skládají ze zpráv příkazů a bloků dat řádků o velikosti vyrovnávací paměti. Ačkoli se podrobnosti mohou lišit v závislosti na síti a počítačích, aktivita databázového serveru probíhá stejným způsobem, jako v jednoduchém modelu s jedním počítačem: *klient* odešle požadavek do jiného počítače, na *server*. Server odpoví blokem dat z tabulky.

Vždy, když dochází k výměně dat prostřednictvím sítě, bude nevyhnutelně docházet k prodlevám v následujících situacích:

- Když je síť zaneprázdněná, musí klient čekat, dokud na něho při přenosu nepřijde řada. Takové prodlevy jsou obvykle menší než jedna milisekunda. Ovšem u velmi zatížených sítí mohou tyto prodlevy narůstat exponenciálně na desítky sekund a více.
- Když server zpracovává požadavky od více než jednoho klienta, požadavky mohou být ve frontě po dobu, která je v rozsahu od milisekund do sekund.
- Když server zpracovává požadavek, zaznamená časovou nákladovost přístupu na disk a operací uvnitř paměti, která byla popsána v předchozích částech.

Přenos odpovědi také způsobuje prodlevy sítě.

Přístupová doba sítě je mimořádně proměnlivá. V nejlepším případě, když nejsou ani síť, ani server zaneprázdněny, jsou prodlevy přenosu a front nepodstatné a server odešle řádek téměř tak rychle, jak by tomu bylo v případě místního serveru. Navíc, pokud klient požádá o druhý řádek, bude stránka pravděpodobně ve vyrovnávacích pamětech serveru.

Naneštěstí spolu se vzrůstající zátěží sítě mají všechny tyto faktory tendenci najednou se zhoršovat. Prodlevy přenosu se zvyšují v obou směrech, což zvětšuje fronty na serveru. Prodleva mezi požadavky snižuje pravděpodobnost toho, že se bude stránka nacházet ve vyrovnávací paměti stránky odpovídajícího počítače. Z těchto důvodů se nákladovost přístupu k síti může změnit náhle a zcela dramaticky.

Pokud u distribuovaného dotazu použijete nejdříve klauzuli SELECT FIRST *n*, budete moci stále vidět pouze požadovaný objem dat. Místní databázový server ovšem neodešle klauzuli SELECT FIRST *n* vzdálenému serveru. Z tohoto důvodu může vzdálený server vrátit více dat.

Optimalizátor, který používá databázový server, předpokládá, že přístup k řádku prostřednictvím sítě trvá déle než přístup k řádku v lokální databázi. Tento odhad zahrnuje nákladovost vyhledání řádku na disku a jeho přenos prostřednictvím sítě.

Další informace o akcích, které by mohly zlepšit výkon v rámci sítě, naleznete v následujících částech:

- “Zvýšení výkonu distribuovaných dotazů” na stránce 13-18
- “Program MaxConnect pro vícenásobná připojení (UNIX)” na stránce 3-23
- “Multiplexní připojení” na stránce 3-22
- “Společné oblasti vyrovnávacích pamětí v síti” na stránce 3-14

Jazyk SQL v rámci rutin SPL

Následující část obsahuje informace o tom, jak a kdy databázový server optimalizuje a provádí příkazy jazyka SQL v rámci rutiny SPL.

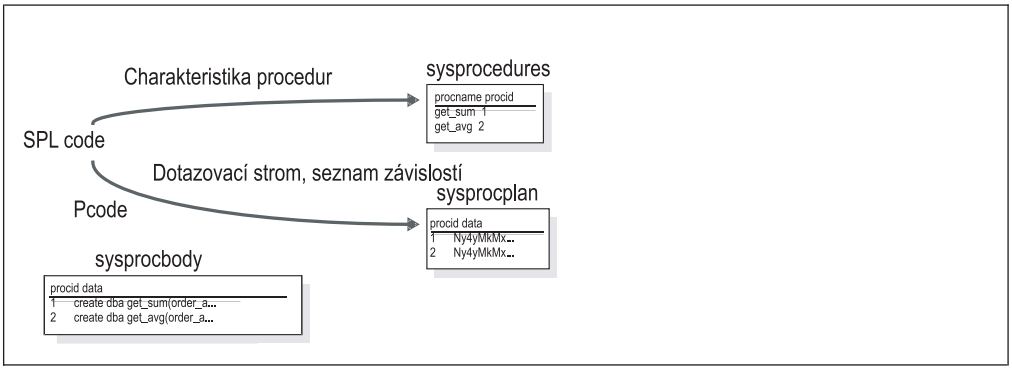
Optimalizace příkazů jazyka SQL

Pokud rutina SPL obsahuje příkazy jazyka SQL, optimalizuje optimalizátor dotazů v určitém bodě možné plány dotazů pro příkazy jazyka SQL v rámci rutiny SPL a vybírá plán dotazů s nejnižší nákladovostí. Databázový server umístí vybraný plán dotazů pro každý příkaz jazyka SQL do plánu provedení rutiny SPL.

Když vytvoříte rutinu SPL s příkazem CREATE PROCEDURE, pokusí se zároveň databázový server optimalizovat příkazy jazyka SQL v rámci rutiny SPL. Pokud není možné zkontrolovat tabulky v čase kompilace (v případě, že neexistují nebo nejsou dostupné), vytváření neselže. V takovém případě databázový server provede optimalizaci příkazů jazyka SQL, které provádí rutina SPL.

Databázový server uloží optimalizovaný plán provedené do tabulky systémového katalogu **sysprocplan**, aby jej mohly používat ostatní procesy. Kromě toho uloží databázový server informace o rutině SPL (například název procedury a jejího vlastníka) do tabulky systémového katalogu **sysprocedures** a verzi ASCII rutiny SPL do tabulky systémového katalogu **sysprocbody**.

Obrázek 10-18 shrnuje informace, které databázový server ukládá do tabulek systémového katalogu během procesu kompilace.



Obrázek 10-18. Informace rutiny SPL uložené v tabulkách systémového katalogu.

Zobrazení plánu provedení

Když provádíte rutinu SPL, je tato rutina již optimalizována. Chcete-li zobrazit plán dotazů pro každý příkaz SQL obsažený v rutině SPL, proveďte příkaz SET EXPLAIN ON předtím, než provedete jeden z následujících příkazů jazyka SQL, pokoušejícího se vždy optimalizovat rutinu SPL:

- CREATE PROCEDURE
- UPDATE STATISTICS FOR PROCEDURE

  Chcete-li například zobrazit plán dotazů pro rutinu SPL, použijte následující dotazy:

```
SET EXPLAIN ON;
UPDATE STATISTICS FOR PROCEDURE název_procedury;
```

Automatická reoptimalizace

Pokud je konfigurační parametr AUTO_REPREPARE proměnné prostředí relace IFX_AUTO_REPREPARE zakázán, může při provedení připravených objektů nebo rutin SPL po modifikaci schématu odkazované tabulky připraveným objektem nebo nepřímo odkazovaného rutinou SPL dojít k následující chybě:

```
-710  Tabulka <název_tabulky> byla vypuštěna, změněna nebo přejmenována.
```

Databázový server používá seznam závislostí ke sledování změn, které by při příštím spuštění rutiny SPL způsobily reoptimalizaci.

Databázový server reoptimalizuje příkaz SQL při příštím spuštění rutiny SPL poté, co se vyskytne jedna z následujících situací:

- Provedení jakéhokoli příkazu jazyka DDL (Data definition language), například ALTER TABLE, DROP INDEX a CREATE INDEX, které mohlo změnit plán dotazů.
- Změna tabulky, která je spojena s jinou tabulkou pomocí referenčního omezení (v obou směrech).
- Provedení příkazu UPDATE STATISTICS FOR TABLE pro libovolnou tabulku zahrnutou do dotazu.

  Příkaz UPDATE STATISTICS FOR TABLE změní číslo verze specifikované tabulky v tabulce **systables.**
- Přejmenování sloupce, databáze nebo indexu pomocí příkazu RENAME.

Kdykoli probíhá reoptimalizace rutiny SPL, používá databázový server tabulku systémového katalogu **sysprocplan** spolu s reoptimalizovaným plánem provedení.

Reoptimalizace rutin SPL

Pokud chcete zabránit vzniku nákladovosti automatické reoptimalizace při prvním provedením rutiny SPL po situacích uvedených v části “Automatická reoptimalizace” na stránce 10-29, proveďte příkaz UPDATE STATISTICS s klauzulí FOR PROCEDURE okamžitě poté, co uvedená situace nastane. Tímto způsobem bude rutina SPL reoptimalizovaná předtím, než ji provedou libovolní uživatelé. Chcete-li zabránit reoptimalizaci všech rutin SPL, která není nezbytná, ujistěte se, že jste do klauzule FOR PROCEDURE zadali specifický název procedury.

```
UPDATE STATISTICS for procedure moje_rutina;
```

Pokyny týkající se příkazu UPDATE STATISTICS naleznete v části “Aktualizace statických údajů, nejsou-li generovány automaticky” na stránce 13-7.

Úrovně optimalizace příkazů SQL v rutinách SPL

Aktuální úroveň optimalizace nastavená v rutině SPL ovlivňuje způsob, jakým je rutina SPL optimalizována.

Algoritmus vyvolaný příkazem SET OPTIMIZATION HIGH představuje sofistikovanou strategii založenou na nákladovosti, která kontroluje všechny rozumné plány dotazů a vybírá všeobecně nejlepší alternativu. V případě velkých spojení může tento algoritmus způsobit větší režii, než je žádoucí. V extrémních případech nemusí dostačovat paměť.

Alternativní algoritmus, který je vyvolán příkazem SET OPTIMIZATION LOW, eliminuje v počátečních stádiích nepravděpodobné strategie spojení, což sníží čas a zdroje spotřebované během optimalizace. Když ovšem zadáte nízkou úroveň optimalizace, nebude možná vybrána optimální strategie, neboť může být vyloučena během počátečních stádií algoritmu.

V případě rutin SPL, které zůstávají nezměněny nebo se změnily pouze málo a které obsahují složité příkazy SELECT, budete možná chtít při vytváření rutiny SPL nastavit příkaz SET OPTIMIZATION na hodnotu HIGH. Tato úroveň optimalizace uloží nejlepší plány dotazů pro rutinu SPL. Potom nastavte optimalizaci na hodnotu LOW předtím, než provedete rutinu SPL. Rutina SPL potom použije optimální plány dotazů a spustí se způsobem, který je nejefektivnější z hlediska nákladovosti, pokud se vyskytne reoptimalizace.

Provedení rutiny SPL

Když databázový server provádí rutinu SPL s příkazem EXECUTE PROCEDURE, příkazem CALL nebo příkazem jazyka SQL, nastanou následující aktivity:

- Databázový server přečte kód interpretu z tabulek systémového katalogu a převede je z komprimovaného formátu do spustitelného formátu. Pokud je rutina SPL v mezipaměti uživatelských rutin, načte ji databázový server z mezipaměti a přeskočí krok převodu.
- Databázový server provede jakékoli příkazy jazyka SPL, které zjistí.
- Když databázový server zjistí příkaz jazyka SQL, vyhledá v databázi plán dotazů a příkaz provede. Pokud nebyl plán dotazů vytvořen, databázový server optimalizuje příkaz jazyka SQL před jeho provedením.
- Poté, co se databázový server dostane na konec rutiny SPL, nebo zaznamená-li příkaz RETURN, vrátí veškeré výsledky klientské aplikaci. Pokud příkaz RETURN neobsahuje klauzuli WITH RESUME, je provedení rutiny SPL dokončeno.

Mezipaměť uživatelských rutin

Když uživatel poprvé provádí rutinu SPL, uloží databázový server spustitelný formát rutiny a všechny plány dotazů do mezipaměti uživatelských rutin (UDR) ve virtuální části sdílené paměti. Pokud další uživatel provádí rutinu SPL, zkontroluje databázový server nejdříve mezipaměť uživatelských rutin. Výkon provedení rutiny SPL se zvýší, pokud může databázový server provádět rutinu SPL z mezipaměti uživatelských rutin. Mezipaměť uživatelských rutin také ukládá uživatelské rutiny, uživatelské souhrny a rozšířené definice datových typů.

Změna mezipaměti uživatelských rutin

Výchozí počet rutin SPL, uživatelských rutin a ostatních uživatelských definic v mezipaměti uživatelských rutin je 127. Tento počet můžete změnit pomocí konfiguračního parametru PC_POOLSIZE.

Databázový server používá k uložení a vyhledávání rutin SPL v mezipaměti uživatelských rutin hashovací algoritmus. Pomocí konfiguračního parametru PC_HASHSIZE můžete změnit počet *sektorů* v mezipaměti uživatelských rutin. Pokud je například hodnota konfiguračního parametru PC_POOLSIZE rovna 100 a PC_HASHSIZE je 10, může mít každý sektor až 10 uživatelských rutin a rutin SPL.

Příliš velký počet sektorů může způsobit, že databázový server přesune rutiny SPL z vyrovnávací paměti, když se sektory zaplní. Příliš malý počet sektorů zvýší počet rutin SPL v sektoru a databázový server musí v sektoru vyhledávat rutiny SPL, aby určil, zda se zde nachází rutina SPL, kterou potřebuje.

Jakmile počet položek v sektoru dosáhne 75 procent, databázový server bude odstraňovat ze sektoru rutiny SPL, které byly použity před nejdelší dobou (a tudíž i z mezipaměti uživatelských rutin, dokud počet rutin SPL v sektoru nedosáhne hodnoty 50 procent z maximálního počtu rutin SPL v sektoru.

Důležité: Konfigurační parametry PC_POOLSIZE a PC_HASHSIZE také řídí další mezipaměti databázového serveru (vyjma společné oblasti paměti): mezipamětpříkazů jazyka SQL, mezipaměť distribuce dat a mezipaměť datového slovníku. Když změníte velikost a počet sektorů hashovací tabulky pro rutiny SPL, změníte také velikost a počet sektorů hashovací tabulky ostatních mezipamětí (například souhrnné mezipaměti, mezipaměti typu oplcass a typename).

Monitorování mezipaměti uživatelských rutin

Chcete-li monitorovat mezipaměť uživatelských rutin, proveďte příkaz **onstat -g prc**. Můžete také provést příkaz **onstat -g cac** a vypsat obsahy ostatních mezipamětí (například souhrnné paměti), stejně jako mezipaměti uživatelských rutin.

Obrázek 10-19 znázorňuje vzorový výstup příkazu **onstat -g prc**.

```
UDR Cache:
    Number of lists            : 31
    PC_POOLSIZE                : 127
UDR Cache Entries:
list#  id  ref_cnt   dropped?   heap_ptr        udr name
--------------------------------------------------------------
0       138     0       0       a4ba820         sales_rep@london:.destroy
3       50      0       0       a4b2020         sales_rep@london:.assign
6       25      0       0       a4b8420         sales_rep@london:.rowoutput
7       29      0       0       a214860         sales_rep@london:.assign
...
```

Obrázek 10-19. Výstup příkazu onstat -g prc.

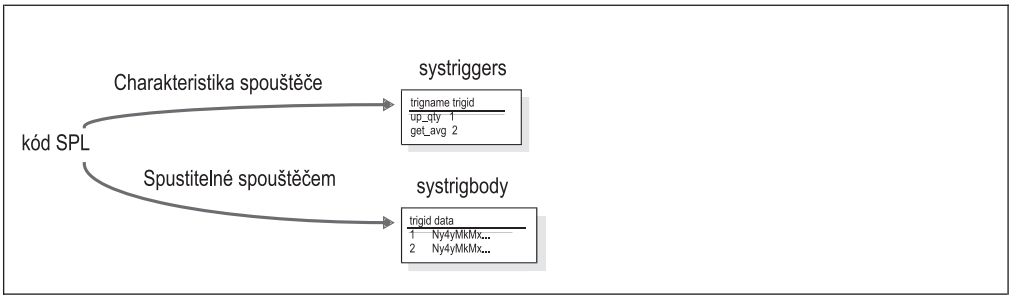
Výstup příkazu **onstat -g prc** obsahuje následující pole:

Pole	Popis
Number of Lists	Počet sektorů specifikovaných konfiguračním parametrem PC_HASHSIZE.
PC_POOLSIZE	Počet uživatelských rutin a SPL povolených v mezipaměti uživatelských rutin.
List #	Počet sektorů.
ID	Jedinečný ID uživatelské rutiny nebo rutiny SPL.
Ref count	Počet relací aktuálně používajících uživatelské rutiny nebo rutiny SPL z mezipaměti.
Dropped	Označení, zda byla rutina SPL označena k vypuštění.
Heap ptr	Ukazatel haldy.
UDR name	Název uživatelské rutiny nebo rutiny SPL.

Provedení spouštěče

Spouštěč je databázový objekt, který automaticky provádí jeden nebo více příkazů jazyka SQL (*akce vyvolaná spouštěčem*), když nastane určená operace jazyka týkající se manipulace s daty (*událost spouštěče*). V tabulce můžete definovat jeden nebo více spouštěčů, které se provedou po událostech spouštěče vyvolaných příkazy SELECT, INSERT, UPDATE nebo DELETE.

U pohledu můžete také definovat spouštěče INSTEAD OF. Tyto spouštěče určují příkazy jazyka SQL, které mají být provedeny jako akce vyvolané spouštěčem v základní tabulce, když se spouštěcí příkazy INSERT, UPDATE nebo DELETE pokusí změnit pohled. Tyto spouštěče jsou označovány jako spouštěče INSTEAD OF, protože je provedena pouze akce jazyka SQL vyvolaná spouštěčem; událost spouštěče provedena není. Další informace o používání spouštěčů naleznete v příručce *IBM Informix Guide to SQL: Tutorial*. Informace o příkazu CREATE TRIGGER uvádí příručka *IBM Informix Guide to SQL: Syntax*.



Obrázek 10-20. Informace spouštěče uložené v tabulkách systémového katalogu.

Při použití příkazu CREATE TRIGGER k zaregistrování nového spouštěče provádí databázový server následující činnosti:

- Uloží informace o spouštěči do tabulky systémového katalogu **systriggers**.
- Uloží text příkazů, které spouštěč provádí, do tabulky systémového katalogu **systrigbody**.

Tabulka systémového katalogu **sysprocedures** identifikuje rutiny spouštěče, které mohou být vyvolány pouze jako akce vyvolané spouštěčem.

Tabulky uložené v paměti databáze **sysmaster** ukazují, zda tabulka nebo pohled obsahují spouštěče.

Kdykoli je vydán příkaz SELECT, INSERT, UPDATE nebo DELETE, kontroluje databázový server, zda je příkaz *událostí spouštěče*, která aktivuje spouštěč pro tuto tabulku a sloupce (příp. pohled), s nimiž pracuje příkaz jazyka DML. Pokud příkaz vyžaduje aktivaci spouštěčů, vyhledá databázový server v tabulce **systrigbody** text příkazu akcí vyvolaných spouštěčem a spustí příkazy jazyka DML vyvolané spouštěčem nebo rutinu SPL předtím, během nebo po událostech spouštěče. V případě spouštěčů INSTEAD OF provede databázový server akce vyvolaná spouštěčem namísto událostí spouštěče.

Vliv spouštěčů na výkon

Spouštěče mohou mírně ovlivnit výkon díky snížení počtu zpráv předávaných od klienta databázovému serveru. Pokud například spouštěč aktivuje pět příkazů jazyka SQL, ušetří klient minimálně deset zpráv předávaných mezi klientem a databázovým serverem (jednu pro odeslání příkazu jazyka SQL a jednu pro odpověď poté, co databázový server příkaz jazyka SQL provede). Spouštěče zvyšují výkon nejvíce, když provádějí více příkazů jazyka SQL a rychlost sítě je relativně pomalá.

Když databázový server provádí příkaz jazyka SQL, musí vykonat následující akce:

- Určit, zda musí být aktivovány spouštěče.
- Vyhledat spouštěče v tabulce systémového katalogu **systriggers** a **systrigbody**.

Tyto operace mají pouze mírný dopad na výkon, který může být vyrovnán sníženým počtem zpráv předaných mezi klientem a databázovým serverem.

U spouštěčů prováděných s příkazy SELECT však dochází k dodatečnému dopadu na výkon. V následujících částech bude tento dopad vysvětlen.

Spouštěče SELECT v tabulkách v hierarchii tabulek

Když databázový server provádí příkaz SELECT zahrnující tabulku, která je součástí hierarchie tabulek, a tento příkaz SELECT aktivuje spouštěč SELECT, může být výkon pomalejší, jestliže příkaz SELECT vyvolávající spouštěč zahrnuje spojení, řazení nebo

materializovaný pohled. V takovém případě databázový server neví, kterých sloupců se příkaz týká, a může proto provést dotaz různým způsobem. Může se vyskytnout následující chování:

- Prohledávání pouze klíčů indexu jsou zakázána v tabulce, která je zahrnuta do hierarchie tabulek.
- Pokud databázový server potřebuje řadit data vybraná z tabulky zahrnuté v hierarchii tabulek, zkopíruje do seznamu SELECT v dočasné tabulce všechny sloupce, ne pouze řazené sloupce.
- Pokud databázový server používá tabulku zahrnutou do hierarchie tabulek k vytvoření hashovací tabulky pro spojení typu hash s jinou tabulkou, obejde prvotní projekci, což znamená, že k vytvoření hashovací tabulky použije všechny sloupce z tabulky, ne pouze sloupce ve spojení.
- Pokud příkazy SELECT obsahují materializovaný pohled (což znamená, že pro sloupce pohledu musí být vytvořena dočasná tabulka), který obsahuje sloupce z tabulky zahrnuté do hierarchie tabulek, budou do dočasné tabulky zahrnuty všechny sloupce tabulky, ne pouze sloupce aktuálně obsažené v pohledu.

Spouštěče SELECT a ukládání řádků do vyrovnávací paměti

V případě příkazů SELECT, jejichž tabulky neaktivují spouštěče SELECT odesílá databázový server více než jeden řádek zpět klientovi a ukládá řádky do vyrovnávací paměti, ačkoli klientská aplikace požadovala prostřednictvím příkazu FETCH pouze jeden řádek. V případě příkazů SELECT, které obsahují jednu nebo více tabulek aktivujících spouštěč SELECT, však databázový server odešle zpět klientovi pouze požadovaný řádek namísto úplného obsahu vyrovnávací paměti. Databázový server však nemůže vrátit ostatní řádky klientovi, dokud nenastane akce spouštěče.

Nedostatečné ukládání do vyrovnávací paměti u příkazů SELECT, které aktivují spouštěče SELECT, může mírně snížit výkon v porovnání s identickými příkazy SELECT, které spouštěče SELECT neaktivují.

Kapitola 11. Direktivy optimalizátoru

Obsah kapitoly

Tato kapitola popisuje *direktivy optimalizátoru*, což jsou komentáře, které optimalizátoru sdělují, jak má provést dotaz. Tato kapitola obsahuje také informace týkající se použití direktiv ke zvýšení výkonu dotazů. Další informace naleznete v částech:

- "Co jsou direktivy optimalizátoru"
- "Důvody k použití direktiv optimalizátoru" na stránce 11-2
- "Příprava na použití direktiv" na stránce 11-3
- "Pokyny týkající se použití direktiv" na stránce 11-4
- "Typy direktiv zahrnutých do příkazů jazyka SQL" na stránce 11-4
- "Konfigurační parametry a proměnné prostředí direktiv optimalizátoru" na stránce 11-11
- "Direktivy optimalizátoru a rutiny SPL" na stránce 11-11
- "Vynucení opětovné optimalizace, která má zabránit problému s indexem a předem vytvořeným příkazem, není-li povolen konfigurační parametr If AUTO_REPREPARE" na stránce 11-12
- "Použití externích direktiv optimalizátoru" na stránce 11-13

Co jsou direktivy optimalizátoru

Direktivy optimalizátoru jsou komentáře, které dávají optimalizátoru dotazů pokyny týkající se způsobu provedení dotazu. Můžete používat dva druhy direktiv optimalizátoru:

- Direktivy optimalizátoru ve formě instrukcí, které jsou vloženy do dotazů
- Externí direktivy optimalizátoru, které můžete vytvořit a uložit pro použití jako dočasná náhradní řešení problémů v případě, že nechcete měnit příkazy jazyka SQL v dotazech.

Další informace naleznete v částech:

- "Direktivy optimalizátoru vložené do dotazů"
- "Externí direktivy optimalizátoru"

Direktivy optimalizátoru vložené do dotazů

Direktivy optimalizátoru jsou komentáře příkazu SELECT, které dávají optimalizátoru dotazů pokyny týkající se způsobu provedení dotazu. Také do příkazů UPDATE a DELETE je možné umístit direktivy, které instruují optimalizátor, jak přistupovat k datům. Direktivy optimalizátoru mohou být buď explicitní pokyny (například "use this index" (použít tento index) nebo "access this table first" (nejprve přistoupit k této tabulce)), nebo mohou eliminovat možné plány dotazů (například "do not read this table sequentially" (tuto tabulku nečíst sekvenčně) nebo "do not perform a nested-loop join" (neprovádět spojení typu vnořená smyčka)).

Externí direktivy optimalizátoru

Externí direktivy optimalizátoru jsou direktivy optimalizátoru, které může administrátor vytvořit a uložit do tabulky katalogu **sysdirectives**. Administrátor může následně tyto direktivy zpřístupnit pomocí proměnné ONCONFIG. Uživatelé klienta také určují proměnnou prostředí a mohou si zvolit, zda budou používat tyto direktivy optimalizátoru v dotazech v situacích, kdy do příkazů jazyka SQL nebudou chtít vkládat komentáře.

Externí direktivy jsou užitečné v případech, kdy není jako krátkodobé řešení problému vhodné přepsat dotaz (například začne-li se dotaz provádět se slabým výkonem). Přepsání dotazu změnou příkazu jazyka SQL se dává přednost při dlouhodobém řešení problémů.

Externí direktivy se používají pouze příležitostně. Počet direktiv uložených v katalogu **sysdirectives** by neměl přesáhnout hodnotu 50. Běžný podnik potřebuje pouze 0 až 9 direktivy.

Důvody k použití direktiv optimalizátoru

Ve většině případů zvolí optimalizátor nejrychlejší plán dotazů. Direktivy optimalizátoru můžete použít v případě, kdy optimalizátor nezvolí k provedení dotazu nejlepší plán dotazů kvůli složitosti dotazu nebo kvůli tomu, že nemá dostatek informací o charakteru dat. Nevhodný plán dotazů poskytuje slabý výkon.

Předtím, než se rozhodnete použít direktivy optimalizátoru, měli byste rozumět tomu, jak pracuje dobrý plán dotazů.

Optimalizátor vytvoří plán dotazů založený na nákladovosti použití různých cest přístupu k tabulce, pořadí spojení a plánů spojení.

Některé pokyny při použití plánů dotazů:

- Nepoužívejte index, pokud databázový server musí číst velkou část tabulky. Například následující dotaz bude možná číst většinu tabulky **customer**:

  ```
  SELECT * FROM customer WHERE STATE <> "ALASKA";
  ```

  Za předpokladu, že jsou zákazníci (z tabulky customer) rovnoměrně rozděleni mezi všech 50 států, můžete odhadovat, že databázový server bude muset číst 98 procent tabulky. Je výhodnější číst tabulku sekvenčně než procházet index (a následně sekvenčně datové stránky), když musí databázový server číst většinu řádků.
- Máte-li si při přístupu k tabulce zvolit mezi indexy, použijte index, který vyloučí většinu řádků. Zvažte například následující dotaz:

```
SELECT * FROM customer
WHERE state = "NEW YORK" AND order_date = "01/20/97"
```

Za předpokladu, že většina 200000 zákazníků (tabulka customer) bydlí v New York a v jednom dni objednalo pouze 1000 zákazníků, zvolí optimalizátor k provedení dotazu pravděpodobně jako index **order_date** (datum objednávky) než index **state** (stát).

- Umístěte malé tabulky nebo tabulky s omezujícími filtry na začátek plánu dotazů. Zvažte například následující dotaz:

```
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
      AND
   customer.state = "NEVADA";
```

V tomto příkladu čtete nejdříve tabulku **customer**, potom můžete vyloučit většinu řádků použitím filtru, který zvolí všechny řádky, ve kterých state = ″NEVADA″.

Vyloučením řádků z tabulky **customer** nečte databázový server tolik řádků v tabulce **orders**, která může být výrazně větší než tabulka **customer**.

- Zvolte spojení typu hash, když ani jeden sloupec ve filtru spojení nemá index.

  Pokud v předchozím příkladu nejsou sloupce **customer.customer_num** a **orders.customer_num** indexovány, bude spojení typu hash pravděpodobně nejlepším plánem spojení.

- Zvolte spojení typu vnořená smyčka, pokud platí:
  - Počet řádek nalezených ve vnější tabulce poté, co databázový server použije veškeré filtry tabulky, je malý a vnitřní tabulka má index, který je možné použít k provedení spojení.
  - Index nejokrajovější tabulky je možné použít k návratu řádků v pořadí klauzule ORDER BY, čímž je vyloučena potřeba řazení.

Další informace týkající se plánů dotazů naleznete v části “Plán dotazů” na stránce 10-2.
Další informace týkající se direktiv naleznete v částech

- “Příprava na použití direktiv”
- “Pokyny týkající se použití direktiv” na stránce 11-4
- “Typy direktiv zahrnutých do příkazů jazyka SQL” na stránce 11-4

Příprava na použití direktiv

Ve většině případů zvolí optimalizátor nejrychlejší plán dotazů. Chcete-li optimalizátoru pomoci a připravit použití direktiv, proveďte následující úlohy:

- Spuštění příkazu UPDATE STATISTICS.

  Bez přesných statistických údajů nemůže optimalizátor zvolit vhodný plán dotazů. Spusťte příkaz UPDATE STATISTICS kdykoli, kdy se data v tabulkách výrazně změní (je přidáno mnoho nových řádků, nebo je mnoho řádků aktualizováno či odstraněno). Další informace naleznete v části “Aktualizace počtu řádků” na stránce 13-8.

- Vytvoření distribucí.

  Jednou z prvních akcí, kterou byste měli vyzkoušet, když se pokoušíte zlepšit výkon pomalého dotazu, je vytvoření distribucí u sloupců zahrnutých do dotazu. Distribuce poskytují optimalizátoru nejpřesnější informace o povaze dat v tabulce. Spusťte příkaz UPDATE STATISTICS HIGH na sloupce zahrnuté do filtrů dotazu a uvidíte, zda se výkon zvýší. Další informace naleznete v části “Vytvoření distribucí dat” na stránce 13-9.

V některých případech optimalizátor dotazů nezvolí nejlepší plán dotazů kvůli složitosti dotazů nebo kvůli tomu, že (dokonce i s distribucemi) nemá dostatek informací o povaze dat. V těchto případech se můžete pokusit zvýšit výkon konkrétního dotazu pomocí direktiv.

Pokyny týkající se použití direktiv

Při použití direktiv mějte na paměti následující pokyny:

- Často přezkušujte efektivitu konkrétní direktivy, abyste se ujistili, že tato direktiva funguje efektivně.
  Představte si dotaz v provozním programu s několika direktivami, které vynucují optimální plán dotazů. Přidání, aktualizace nebo odstranění velkého počtu řádků provedené před několika dny uživateli změnilo povahu dat do takové míry, že optimální plán dotazů není nadále efektivní. Tento příklad ilustruje, že je nezbytné používat direktivy opatrně.
- Kdekoli je to možné, používejte negativní direktivy (AVOID_NL, AVOID_FULL, apod.). Když vyloučíte chování snižující výkon můžete se spolehnout na to, že optimalizátor použije následující nejlepší volbu, místo toho abyste vynucovali cestu, která nemusí být optimální.

Typy direktiv zahrnutých do příkazů jazyka SQL

Tato část obsahuje informace o direktivách, které jsou vloženy do dotazů.

Direktivy se zahrnují do příkazů jazyka SQL jako komentář, který následuje bezprostředně po klíčovém slovu SELECT, UPDATE nebo DELETE. Prvním znakem v direktivě je vždy znaménko plus (+). V následujícím dotazu určuje direktiva ORDERED, že se mají tabulky spojit ve stejném pořadí, v jakém jsou uvedeny v klauzuli FROM. Direktiva AVOID_FULL určuje, že by optimalizátor měl vyřadit všechny výsledky úplného prohledávání tabulek v uvedené tabulce (**employee** (zaměstnanec)).

```
SELECT --+ORDERED, AVOID_FULL(e) * FROM employee e, department d
> 50000;
```

Úplný popis syntaxe direktiv naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Volba plánu dotazů prováděná optimalizátorem může být změněna pomocí jednoho z následujících aspektů dotazu:

- Přístupová metoda
- Pořadí spojení
- Plán spojení
- Cíl optimalizace

Můžete také pomocí direktivy EXPLAIN (namísto příkazu SET EXPLAIN) zobrazit plán dotazů. Následující části popisují uvedené aspekty podrobně.

Direktivy přístupové metody

Databázový server používá k přístupu k tabulce přístupovou metodu. Server může číst tabulku buď sekvenčně (prostřednictvím úplného prohledávání tabulky), nebo použít u tabulky jeden z indexů. Následující direktivy ovlivňují přístupovou metodu:

INDEX
Sděluje optimalizátoru, aby k přístupu k tabulce použil určený index. Pokud direktiva uvádí více než jeden index, zvolí si optimalizátor index s nejnižší nákladovostí.

AVOID_INDEX
Sděluje optimalizátoru, aby nepoužíval žádný z uvedených indexů. Tuto direktivu je možné použít s direktivou AVOID_FULL.

INDEX_SJ
Vynutí cestu k indexu spojení typu self-join za použití určeného indexu nebo

zvolením nejméně nákladného indexu ze seznamu indexů a to i tehdy, když nejsou k dispozici statistické údaje distribuce dat pro klíčové sloupce s hlavním indexem tohoto indexu. Informace o cestách k indexu spojení typu self-join naleznete v části “Plány dotazů zahrnující cestu k indexu spojení typu self-join” na stránce 10-8.

AVOID_INDEX_SJ
Sděluje optimalizátoru, aby nepoužíval cestu k indexu spojení typu self-join pro určený index nebo indexy.

FULL Sděluje optimalizátoru, aby provedl úplné prohledávání tabulek.

AVOID_FULL
Sděluje optimalizátoru, aby u uvedené tabulky neprováděl úplné prohledávání tabulky. Tuto direktivu je možné použít s direktivou AVOID_INDEX.

V některých případech může vynucení přístupové cesty změnit metodu spojení, kterou si optimalizátor zvolí. Pokud například vyloučíte použití indexu pomocí direktivy AVOID_INDEX, může si optimalizátor zvolit spojení typu hash namísto spojení typu vnořená smyčka.

Optimalizátor bude uvažovat o cestě k indexu spojení typu self-join, pokud budou splněny všechny následující podmínky:

- Index nemá funkční klíče, typy definované uživatelem, vestavěné netransparentní typy nebo indexy jiné než indexy stromu-B.
- Pro posuzovaný klíčový sloupec s indexem jsou k dispozici statistické údaje distribuce dat.
- Počet řádků v tabulce je roven minimálně desetinásobku jedinečných kombinací všech možných hodnot sloupce s hlavním klíčem.

Pokud jsou splněny všechny tyto podmínky, odhadne optimalizátor nákladovost cesty k indexu spojení typu self-join a porovná ji s nákladovostí alternativních přístupových metod. Poté optimalizátor vybere nejlepší přístupovou metodu pro tabulku. Další informace o direktivách INDEX_SJ a AVOID_INDEX_SJ a řadu příkladů použití direktivy INDEX_SJ naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Direktivy pořadí spojení

Direktiva pořadí spojení ORDERED přinutí optimalizátor spojit tabulky v pořadí, ve kterém je uvádí příkaz SELECT.

Vliv pořadí spojení na plán spojení

Zadáním pořadí spojení můžete ovlivnit více než jen způsob spojení tabulek. Zvažte například následující dotaz:

```
SELECT --+ORDERED, AVOID_FULL(e)
* FROM employee e, department d
WHERE e.dept_no = d.dept_no AND e.salary > 5000
```

V tomto příkladu optimalizátor zvolí spojit tabulky spojením typu hash. Pokud ovšem uspořádáte pořadí tak, že druhá tabulka bude **employee**(zaměstnanec), ke které se musí přistupovat pomocí indexu, spojení typu hash nebude proveditelné.

```
SELECT --+ORDERED, AVOID_FULL(e)
* FROM department d, employee e
WHERE e.dept_no = d.dept_no AND e.salary > 5000;
```

V tomto případě zvolí optimalizátor spojení typu vnořená smyčka.

Pořadí spojení při použití pohledů

Pořadí spojení při použití pohledů mohou ovlivnit dva faktory:

- Direktiva ORDERED je uvnitř pohledu.

  Direktiva ORDERED uvnitř pohledu ovlivňuje pořadí spojení pouze tabulek, které jsou uvnitř pohledu. Tabulky uvnitř pohledu musí být spojeny po sobě za sebou. Zvažte například následující pohled a dotaz:

```
CREATE VIEW emp_job_view as
   SELECT {+ORDERED}
   emp.job_num, job.job_name
   FROM emp, job
   WHERE emp.job_num = job.job_num;

SELECT * from dept, emp_job_view,  project
   WHERE dept.dept_no = project.dept_num
   AND emp_job_view.job_num = project.job_num;
```

  Direktiva ORDERED určuje, že tabulka **emp** předchází tabulku job. Tato direktiva neovlivňuje pořadí tabulek **dept** a **project**. Z tohoto důvodu existují následující možná spojení:

  - **emp**, **job**, **dept**, **project**
  - **emp**, **job**, **project**, **dept**
  - **project**, **emp**, **job**, **dept**
  - **dept**, **emp**, **job**, **project**
  - **dept**, **project**, **emp**, **job**
  - **project**, **dept**, **emp**, **job**

- Direktiva ORDERED je v dotazu, který obsahuje více pohledů.

  Pokud se v dotazu obsahujícímu pohled vyskytne direktiva ORDERED, pořadí spojení tabulek odpovídá pořadí, ve kterém jsou uvedeny v příkazu SELECT. Tabulky v rámci pohledu jsou spojeny tak, jak jsou uvedeny v pohledu.

  V následujícím dotazu je pořadí spojení **dept**, **project**, **emp**, **job**:

```
CREATE VIEW emp_job_view AS
   SELECT
   emp.job_num, job.job_name
   FROM emp, job
   WHERE emp.job_num = job.job_num;
SELECT {+ORDERED}
   * FROM dept, project, emp_job_view
   WHERE dept.dept_no = project.dept_num
   AND emp_job_view.job_num = project.job_num;
```

  Výjimkou z tohoto pravidla je situace, kdy pohled nelze složit do dotazu, jak uvádí následující příklad:

```
CREATE VIEW emp_job_view2 AS
   SELECT DISTINCT
   emp.job_num, job.job_name
   FROM emp,job
   WHERE emp.job_num = job.job_num;
```

  V tomto příkladu provede databázový server dotaz a vloží výsledek do dočasné tabulky. Pořadí tabulek v dotazu je **dept**, **project**, **temp_table**.

Direktivy plánu spojení

Direktivy plánu spojení ovlivňují způsob, jakým databázový server spojí dvě tabulky v dotazu.

Následující direktivy ovlivňují plán spojení mezi dvěma tabulkami:

- USE_NL

Použít uvedenou tabulku ve spojení typu vnořená smyčka.

- USE_HASH

  Přistupovat k uvedeným tabulkám pomocí spojení typu hash. Můžete si také zvolit, zda bude tabulka použita k vytvoření hashovací tabulky nebo k prohledání hashovací tabulky.

- AVOID_NL

  Nepoužívat uvedené tabulky jako vnitřní tabulku ve spojení typu vnořená smyčka. Tabulka uvedena v této direktivě se však může účastnit spojení typu vnořená smyčka jako vnější tabulka.

- AVOID_HASH

  Nepřistupovat k uvedeným tabulkám pomocí spojení typu hash. Volitelně je možné povolit spojení typu hash, ale zamezte tomu, aby tabulka nebyla prohledávanou tabulkou, ze které je spojení typu hash vytvářeno.

Direktivy cílů optimalizace

V některých dotazech budete možná chtít nalézt ve výsledku dotazu pouze několik prvních řádků (například program ESQL/C otevře pro dotaz kurzor a provede příkaz FETCH k nalezení pouze prvního řádku). Nebo můžete vědět, že musí být přistupováno ke všem řádkům a všechny řádky musí být vráceny. Direktivy optimalizace dotazu můžete použít k optimalizaci dotazu v jednom z následujících případů:

- FIRST_ROWS

  Zvolení plánu, který optimalizuje proces nalezení pouze prvního řádku vyhovujícího dotazu.

- ALL_ROWS

  Zvolení plánu, který optimalizuje proces nalezení všech řádků (výchozí chování) vyhovujících dotazu.

Pokud použijete direktivu FIRST_ROWS, může optimalizátor vypustit plán dotazů obsahujícího aktivity, které jsou zpočátku časově náročné. Například spojení typu hash může trvat dlouhou dobu, než vytvoří hashovací tabulku. Pokud musí být vráceno pouze několik řádků, může místo toho optimalizátor zvolit spojení typu vnořená smyčka.

V následujícím příkladu předpokládejme, že má databáze index ve sloupci **employee.dept_no**, ale nemá jej ve sloupci **department.dept_no**. Bez použití direktiv by optimalizátor zvolil spojení typu hash.

```
SELECT *
FROM employee, department
WHERE employee.dept_no = department.dept_no
```

Ovšem s direktivou FIRST_ROWS zvolí optimalizátor spojení typu vnořená smyčka kvůli vysoké počáteční nákladovosti požadované na vytvoření hashovací tabulky.

```
SELECT {+first_rows} *
FROM employee, department
WHERE employee.dept_no = department.dept_no
```

Příklady použití direktiv

Následující příklad znázorňuje, jak mohou direktivy změnit plán dotazů.

Předpokládejme, že máte následující dotaz:

```
SELECT * FROM emp,job,dept
WHERE emp.location = 10
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER";
```

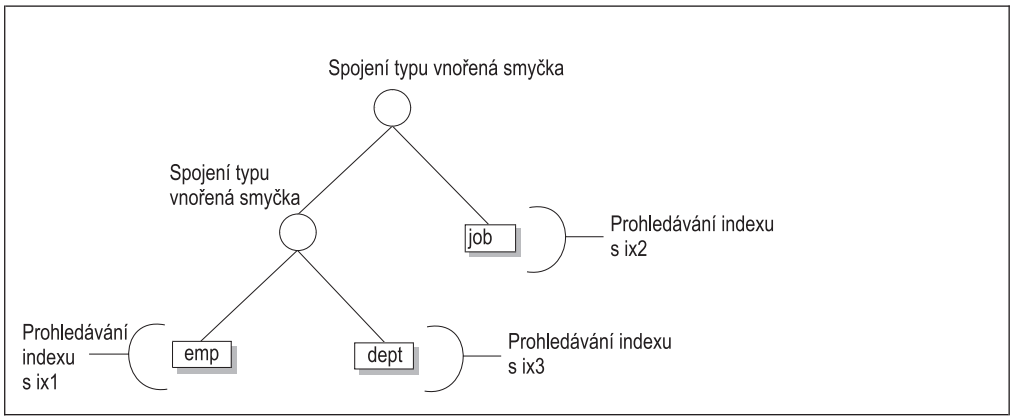
Předpokládejme, že existují následující indexy:

```
ix1: emp(empno,jobno,deptno,location)
ix2: job(jobno)
ix3: dept(location)
```

Chcete-li zobrazit cestu dotazu zvolenou optimalizátorem, můžete spustit dotaz s příkazem SET EXPLAIN ON.

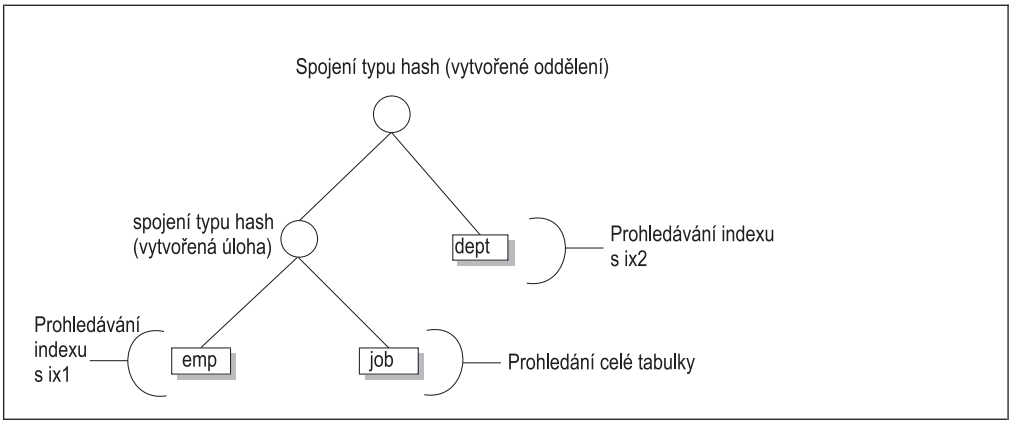
```
QUERY:
------
SELECT * FROM emp,job,dept
WHERE emp.location = "DENVER"
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER"

Estimated Cost: 5
Estimated # of Rows Returned: 1

1) informix.emp: INDEX PATH

    Filters: informix.emp.location = 'DENVER'

    (1) Index Keys: empno jobno deptno location   (Key-Only)

2) informix.dept: INDEX PATH

    Filters: informix.dept.deptno = informix.emp.deptno

    (1) Index Keys: location
        Lower Index Filter: informix.dept.location = 'DENVER'
NESTED LOOP JOIN

3) informix.job: INDEX PATH

    (1) Index Keys: jobno   (Key-Only)
        Lower Index Filter: informix.job.jobno = informix.emp.jobno
NESTED LOOP JOIN
```

Diagram, který uvádí Obrázek 11-1, znázorňuje možné plány dotazů pro tento dotaz.



Obrázek 11-1. Možný plán dotazů bez direktiv.

Možná se domníváte, že použití spojení typu vnořená smyčka nemusí být nejrychlejší metodou k provedení tohoto dotazu. Rovněž si myslíte, že pořadí spojení není optimální. Můžete přinutit optimalizátor, aby zvolil spojení typu hash a seřadil tabulky v plánu dotazů podle jejich pořadí v dotazu. Optimalizátor potom použije plán dotazů, který znázorňuje Obrázek 11-2.



Obrázek 11-2. Možný plán dotazů s direktivami.

Chcete-li vynutit, aby optimalizátor zvolil plán dotazů, který používá spojení typu hash a pořadí tabulek uvedené v dotazu, použijte direktivy tak, jak je uvedeno v částečném výstupu příkazu SET EXPLAIN:

```
QUERY:
------
SELECT {+ORDERED,
   INDEX(emp ix1),
   FULL(job),
   USE_HASH(job /BUILD),
   USE_HASH(dept /BUILD),
   INDEX(dept ix3)}
   * FROM emp,job,dept
   WHERE emp.location = 1
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER"

DIRECTIVES FOLLOWED:
ORDERED
INDEX ( emp ix1 )
FULL ( job )
USE_HASH ( job/BUILD )
USE_HASH ( dept/BUILD )
INDEX ( dept ix3 )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 1

1) informix.emp: INDEX PATH

    Filters: informix.emp.location = 'DENVER'

    (1) Index Keys: empno jobno deptno location   (Key-Only)

2) informix.job: SEQUENTIAL SCAN


DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.emp.jobno = informix.job.jobno

3) informix.dept: INDEX PATH

    (1) Index Keys: location
        Lower Index Filter: informix.dept.location = 'DENVER'

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.emp.deptno = informix.dept.deptno
```

Direktivy EXPLAIN

Pomocí direktiv EXPLAIN je možné zobrazit plán dotazů následujícími způsoby:

- EXPLAIN

  Zobrazení plánu dotazů, který si zvolí optimalizátor.
- EXPLAIN AVOID_EXECUTE

  Zobrazení plánu dotazů, který si zvolí optimalizátor, bez provedení dotazu.

Pokud chcete zobrazit plán dotazů pouze pro jeden příkaz jazyka SQL, použijte tyto direktivy EXPLAIN namísto příkazů SET EXPLAIN ON nebo SET EXPLAIN ON AVOID_EXECUTE.

Pokud použijete AVOID_EXECUTE (ať již v direktivě, nebo v příkazu SET EXPLAIN), dotaz se neprovede, zobrazí však následující zpráva:

```
No rows returned. (Nejsou vráceny žádné řádky).
```

Obrázek 11-3 znázorňuje vzorový výstup pro dotaz, který používá direktivu EXPLAIN AVOID_EXECUTE.

```
QUERY:
------
select --+ explain avoid_execute
  l.customer_num, l.lname, l.company,
  l.phone, r.call_dtime, r.call_descr
from customer l, cust_calls r
where l.customer_num = r.customer_num

DIRECTIVES FOLLOWED:
EXPLAIN
AVOID_EXECUTE
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 7

  1) informix.r: SEQUENTIAL SCAN

  2) informix.l: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.l.customer_num = informix.r.customer_num
NESTED LOOP JOIN
```

Obrázek 11-3. Výsledek direktiv EXPLAIN AVOID_EXECUTE.

V následující tabulce jsou popsány příslušné řádky výstupu popisující zvolený plán dotazů, který znázorňuje Obrázek 11-3.

Řádek výstupu, který znázorňuje Obrázek 11-3	Popis zvoleného plánu dotazů
DIRECTIVES FOLLOWED: EXPLAIN AVOID_EXECUTE	Použití direktivy EXPLAIN a AVOID_EXECUTE k zobrazení plánu dotazů a k neprovedení dotazu.
Estimated # of Rows Returned: 7	Odhad, že tento dotaz vrátí sedm řádků.
Estimated Cost: 7	Odhadovaná nákladovost hodnoty 7 je hodnota, kterou optimalizátor používá k porovnání různých plánů dotazů a výběru jednoho z nich, který má nejnižší nákladovost.
1) informix.r: SEQUENTIAL SCAN	Použití tabulky **cust_calls r** jako vnější tabulky a její prohledávání za účelem získání každého řádku.
2) informix.l: INDEX PATH	Použití indexu pro každý řádek vnější tabulky za účelem získání shodných řádků ve vnitřní tabulce **customer l**.
(1) Index Keys: customer_num (Serial, fragments: ALL)	Použití indexu ve sloupci **customer_num**, jeho sériové prohledávání a prohledávání všech fragmentů (tabulka **customer l** se skládá pouze z jednoho fragmentu).
Lower Index Filter: informix.l.customer_num = informix.r.customer_num	Spuštění prohledávání indexu na hodnotě **customer_num** z vnější tabulky.

Konfigurační parametry a proměnné prostředí direktiv optimalizátoru

Konfigurační parametr DIRECTIVES je možné použít k zapnutí nebo vypnutí všech direktiv, se kterými se databázový server setká. Pokud má konfigurační parametr DIRECTIVES hodnotu 1 (výchozí hodnota), optimalizátor se řídí všemi direktivami. Pokud má konfigurační parametr DIRECTIVES hodnotu 0, optimalizátor ignoruje všechny direktivy.

Nastavení konfiguračního parametru DIRECTIVES můžete přepsat pomocí proměnné prostředí **IFX_DIRECTIVES**. Je-li proměnná prostředí **IFX_DIRECTIVES** nastavena na hodnotu 1 nebo ON, řídí se optimalizátor direktivami pro každý příkaz jazyka SQL, který provádí relace klienta. Pokud je proměnná prostředí **IFX_DIRECTIVES** nastavena na hodnotu 0 nebo OFF, ignoruje optimalizátor direktivy jakéhokoli příkazu jazyka SQL v relaci klienta.

Jakékoli direktivy v příkazu jazyka SQL mají přednost před plánem spojení, který je vynucen konfiguračním parametrem OPTCOMPIND. Pokud například dotaz obsahuje direktivu USE_HASH a konfigurační parametr OPTCOMPIND je nastaven na hodnotu 0 (spojení typu vnořená smyčka mají přednost před spojeními typu hash), použije optimalizátor spojení typu hash.

Direktivy optimalizátoru a rutiny SPL

V dotazu v rutině SPL fungují rutiny odlišně, protože příkaz SELECT v rutině SPL není nezbytně optimalizován okamžitě předtím, než jej databázový server provede. Optimalizátor vytváří plán dotazů pro příkaz SELECT v rutině SPL poté, co databázový server vytvoří rutinu SPL, nebo během provedení některých verzí příkazu UPDATE STATISTICS.

Optimalizátor čte a používá direktivy v době vytváření plánu dotazů. Protože databázový server ukládá plán dotazů do tabulky systémového katalogu, není příkaz SELECT při provádění reoptimalizován. Z tohoto důvodu nastavené hodnoty konfiguračních parametrů

IFX_DIRECTIVES a DIRECTIVES ovlivňují příkazy SELECT uvnitř rutiny SPL, pokud jsou nastaveny v jednom z níže uvedených okamžiků:

- Před příkazem CREATE PROCEDURE
- Před příkazem UPDATE STATISTICS, který způsobí optimalizaci příkazu jazyka SQL v rutině SPL
- Za určitých okolností, pokud jsou příkazům SELECT dodávány proměnné za běhu programu

Vynucení opětovné optimalizace, která má zabránit problému s indexem a předem vytvořeným příkazem, není-li povolen konfigurační parametr If AUTO_REPREPARE

Pokud je povolen konfigurační parametr AUTO_REPREPARE a proměnná prostředí relace IFX_AUTO_REPREPARE, server Dynamic Server automaticky znovu zkompiluje připravené příkazy a rutiny SPL, jakmile se schéma odkazované tabulky změní příkazem DDL. Informace v této části jsou určeny pouze pro případ, že jsou konfigurační parametr AUTO_REPREPARE a proměnná prostředí relace IFX_AUTO_REPREPARE zakázány.

Pokud jsou konfigurační parametr AUTO_REPREPARE nebo proměnná prostředí relace IFX_AUTO_REPREPARE zakázány, může dojít k následující chybě při provádění připravených objektů nebo rutin SPL, jakmile se změní schéma nebo tabulka odkazované připraveným objektem nebo nepřímo odkazované rutinou SPL.

```
-710 Tabulka <název_tabulky> byla vypuštěna, změněna nebo přejmenována.
```

Tato chyba se může vyskytovat u explicitně připravených příkazů. Tyto příkazy mají následující formát:

```
PREPARE statement id FROM quoted string
```

Po připravení příkazu na databázovém serveru a před jeho provedením mohla být tabulka, na kterou se příkaz odkazuje, přejmenována nebo změněna, případně byla změněna její struktura. V důsledku toho mohou nastat problémy.

Přidání indexu do tabulky poté, co byl příkaz připraven, může také zrušit platnost příkazu. Následný příkaz OPEN pro kurzor se nezdaří, pokud se kurzor odkazuje na neplatný připravený příkaz. K selhání dochází dokonce i tehdy, pokud příkaz OPEN obsahuje klauzuli WITH REOPTIMIZATION.

Pokud byl index přidán po připravení příkazu, musíte připravit příkaz znovu přiřadit a znovu deklarovat kurzor. Nelze jednoduše znovu otevřít kurzor, pokud byl založen na již neplatném připraveném příkazu.

Tato chyba může také nastat u rutin SPL. Předtím, než databázový server poprvé provede novou rutinu SPL, optimalizuje kód (příkazy) v této rutině SPL. Optimalizace způsobí, že se kód stane závislý na struktuře tabulky, na kterou se procedura odkazuje. Pokud se struktura tabulky změní po optimalizaci, ale před provedením této procedury, může se vyskytnout uvedená chyba.

Každá rutina SPL je optimalizována při prvním spuštění (ne v okamžiku vytvoření). Toto chování znamená, že rutina SPL může poprvé úspěšně proběhnout, ale může selhat později za virtuálně identických okolností. Selhání rutiny SPL může být občasné, protože selhání během jednoho provedení vynutí vnitřní upozornění a reoptimalizaci procedury před příštím provedením.

Databázový server uchovává seznam tabulek, na které se rutina SPL explicitně odkazuje. Pokaždé, když je změněna jakákoli z těchto tabulek, na které se rutina explicitně dokazuje, reoptimalizuje databázový server tuto proceduru při jejím příštím provedení.

Pokud ovšem rutina SPL závisí na tabulce, která na kterou je odkazováno pouze nepřímo, nemůže databázový server po změně této tabulky zjistit potřebu reoptimalizace procedury. Odkaz na tabulku může být nepřímý například tehdy, pokud rutina SPL vyvolá spouštěč. Jestliže je změněna tabulka, na kterou se odkazuje spouštěč (a ne přímo rutina SPL), databázový server neví, že by měl rutinu SPL před jejím spuštěním reoptimalizovat. Když je procedura spuštěna po změně této tabulky, může se vyskytnout uvedená chyba.

K odhalení uvedené chyby použijte jednu z následujících dvou metod:

- Vydání příkazu UPDATE STATISTICS za účelem vynucení reoptimalizace procedury.
- Opakované spuštění procedury.

Chcete-li uvedené chybě zabránit, můžete vynutit reoptimalizaci rutiny SPL. Při vynucení reoptimalizace spusťte následující příkaz:

```
UPDATE STATISTICS FOR PROCEDURE procedure name
```

Tento příkaz je možné přidat k programu jedním z následujících způsobů:

- Umístěte příkaz UPDATE STATISTICS za každý příkaz, který změní režim objektu.
- Umístěte příkaz UPDATE STATISTICS před každé provedení rutiny SPL.

Z důvodů efektivity můžete umístit příkaz UPDATE STATISTICS u akce, která se v programu vyskytuje méně často (změna režimu objektu nebo provedení procedury). Ve většině případů se v programu vyskytuje méně často změna režimu objektu.

Když se řídíte uvedenou metodou obnovení z této chyby, musíte provést příkaz UPDATE STATISTICS pro každou proceduru, která se odkazuje na změněné tabulky nepřímo, pokud se na tyto tabulky neodkazuje procedura také přímo.

Obnovu z této chyby je také možné provést jednoduše opakovaným spuštěním rutiny SPL. Při prvním selhání procedury ji databázový server označí jako proceduru, která vyžaduje reoptimalizaci. Při příštím spuštění této procedury ji databázový server před jejím spuštěním reoptimalizuje. Dvojí spouštění rutiny SPL však nemusí být praktické, ani bezpečné. Bezpečnější volbou je použít příkaz UPDATE STATISTICS k vynucení reoptimalizace procedury.

Použití externích direktiv optimalizátoru

Pokud jste uživatel **informix**, můžete vytvářet, ukládat a odstraňovat externí direktivy.

Vytvoření a uložení externích direktiv

Postup vytvoření a uložení externích direktiv:

1. Vytvořte *záznamy přidružení*, které použijete pro direktivy dotazů.

   Záznamy přidružení jsou záznamy, které může administrátor vložit do příkazu jazyka SQL, je-li to potřeba. Následující příklad uvádí řádek záznamu přidružení, který je možné použít jako direktivu dotazu.

   ```
   save external directives {+INDEX(t1,i11)} active for
   select {+INDEX(t1, i2) } c1 from t1 where c1=1;

   id            16
   query         select {+INDEX(t1, i2) } c1 from t1 where c1=1
   directive     INDEX(t1,i11)
   directivecode BYTE value
   ```

```
active          1
hashcode        -589336273
```

2. Uložte direktivy do katalogu **sysdirectives** pomocí příkazu SAVE EXTERNAL DIRECTIVES, jak je uvedeno v následujícím příkladu:

```
SAVE EXTERNAL DIRECTIVES directives [ACTIVE/INACTIVE/TEST ONLY]
FOR Select/* INDEX( table1 , index1 )*/ col1 , col2
From table1, table 2
Where table1.col1+table2.col1
```

   Informace uvedená v hodnotě *directives* musí být ve stejném formátu v rámci komentáře přesně tak, jak by se zobrazily direktivy v příkazech SELECT, UPDATE a DELETE. (Další informace o příkazu SAVE EXTERNAL DIRECTIVES naleznete v příručce *IBM Informix Guide to SQL: Syntax*.)

Povolení externích direktiv

Poté, co externí direktivy vytvoříte a uložíte, je třeba nastavit proměnné prostředí, které tyto direktivy povolí. Databázový server vyhledává direktivy pro dotaz pouze tehdy, pokud jsou externí direktivy nastaveny na databázovém serveru i klientovi.

Direktivu je možné povolit pomocí kombinace konfiguračního parametru EXT_DIRECTIVES, který se nachází v souboru ONCONFIG, a proměnné prostředí **IFX_EXTDIRECTIVES** na straně klienta.

Hodnoty konfiguračního parametru EXT_DIRECTIVE, které můžete použít:

Hodnota	Popis
0 (výchozí hodnota)	Vypnuto. Direktivu nelze povolit, i když je proměnná prostředí IFX_EXTDIRECTIVES povolena.
1	Zapnuto. Direktivu je možné pro relaci povolit, pokud je proměnná prostředí IFX_EXTDIRECTIVES povolena.
2	Zapnuto. Direktivu je možné použít, i když není proměnná prostředí IFX_EXTDIRECTIVES povolena.

Soubor sqexplain.out, který je popsaný v části “Soubor sqexplain.out” na stránce 10-10, informuje, zda jsou externí direktivy platné.

Odstranění externích direktiv

Pokud již nadále direktivu nepotřebujete, můžete ji pomocí příkazů jazyka SQL odstranit z katalogu **sysdirectives**.

Kapitola 12. Paralelní databázový dotaz

Obsah kapitoly

Tato kapitola popisuje parametry a strategie, které používáte při správě zdrojů pro paralelní databázový dotaz.

Co je paralelní databázový dotaz (PDQ)

PDQ (Parallel database query) je funkce databázového serveru Informix, která může výrazně zvýšit výkon při zpracování dotazů, které vyvolávají aplikace podporující rozhodování. Funkce PDQ umožňuje databázovému serveru rozdělit zpracování jednoho dotazu na více procesorů. Vyžaduje-li například dotaz agregaci, může databázový server rozdělit zpracování agregace mezi několik procesorů. Funkce PDQ také obsahuje nástroje pro správu zdrojů.

Další funkce databázového serveru, *fragmentace tabulek*, umožňuje ukládat různé části tabulek na různé disky. Funkce PDQ přináší maximální výkon, jsou-li dotazovaná data ve fragmentovaných tabulkách. Informace o tom, jak pomocí fragmentace dosáhnout maximálního výkonu naleznete v části “Naplánování strategie fragmentace” na stránce 9-2.

Struktura funkce PDQ

Každý dotaz pro podporu rozhodování má primární jednotkový proces. Databázový server může spuštěním dalších jednotkových procesů provádět úlohy pro tohoto dotazu (například prohledávání a řazení). V závislosti na počtu tabulek nebo fragmentů, které musí dotaz vyhledat, a na zdrojích, které jsou pro dotaz pro podporu rozhodování dostupné, přiřadí databázový server různým komponentám dotazu různé jednotkové procesy. Databázový server tyto jednotkové procesy PDQ, které jsou ve výstupu SET EXPLAIN uvedeny jako *sekundární jednotkové procesy*, vyvolá.

Sekundární jednotkové procesy jsou podle své funkce dále klasifikovány buď jako *producenti*, nebo jako *konzumenti*. Jednotkový proces producenta dodává data jinému jednotkovému procesu. Jednotkový proces prohledávání by mohl například číst data ze sdílené paměti, která odpovídá dané tabulce, a předat je jednotkovému procesu spojení. V tomto případě je jednotkový proces prohledávání považován za producenta a jednotkový proces spojení je považován za konzumenta. Jednotkový proces spojení by potom mohl data předat jednotkovému procesu řazení. V tomto případě je jednotkový proces spojení považován za producenta a jednotkový proces řazení za konzumenta.

Několik producentů může předávat data jednomu konzumentovi. V tomto případě nastaví databázový server vnitřní mechanismus zvaný *výměna*, který synchronizuje přenos dat od těchto producentů ke konzumentovi. Chcete-li například seřadit fragmentovanou tabulku, optimalizátor obvykle vyvolá samostatný jednotkový proces prohledávání pro každý fragment. Lze očekávat, že jednotkové procesy řazení budou dokončeny v různých časech, protože charakteristiky vstupu - výstupu jsou různé. Pomocí výměny jsou data vytvořená různými jednotkovými procesy řazení předávána do jednoho nebo více jednotkových procesů s ukládáním do vyrovnávací paměti. Podle složitosti dotazu by optimalizátor mohl vyvolat hierarchii producentů, výměn a konzumentů s několika vrstvami. Obecně řečeno: Jednotkové procesy konzumentů a jednotkové procesy producentů pracují současně, takže objem ukládání do vyrovnávací paměti, které výměny provádějí, je zanedbatelný.

Databázový server vytváří tyto jednotkové procesy a výměny automaticky a transparentně. Jsou automaticky ukončeny, jakmile dokončí zpracování daného dotazu. Databázový server vytváří nové jednotkové procesy a výměny pro následné dotazy podle potřeby.

Operace databázového serveru, které používají PDQ

Tato část popisuje dva typy operací SQL, které databázový server současně zpracovává, a situace, které omezují stupeň tohoto paralelismu, který může databázový server použít. V následující diskusi je *dotaz* libovolný příkaz SELECT.

Paralelní odstranění

Při zpracování příkazů DELETE, INSERT a UPDATE provádí databázový server tyto dva kroky:

1. Načte příslušné řádky.
2. Použije akci odstranění, vkládání nebo aktualizace.

Databázový server provádí současně první krok příkazu DELETE s jednou výjimkou: Má-li cílová tabulka referenční omezení, které může přecházet do podřízené tabulky, databázový server neprovádí první krok příkazu DELETE současně.

Paralelní vkládání

Databázový server provádí současně následující typy vkládání:

- Příkaz SELECT...INTO TEMP vkládá pomocí explicitních dočasných tabulek.
- Příkaz INSERT INTO...SELECT vkládá pomocí implicitních dočasných tabulek.

Informace o implicitních a explicitních dočasných tabulkách naleznete v kapitole o ukládání dat v příručce *Příručka administrátora serveru IBM Informix Dynamic Server*.

Explicitní vkládání příkazem SELECT...INTO TEMP

Databázový server může současně vkládat řádky do explicitních dočasných tabulek, které zadáte pomocí příkazů SQL ve tvaru SELECT....INTO TEMP. Může například současně provést vkládání do dočasné tabulky **temp_table**, jak ukazuje následující příklad:

```
SELECT * FROM table1 INTO TEMP temp_table
```

Postup při paralelním vkládání do dočasné tabulky:

1. Nastavte prioritu PDQ na hodnotu 0.

   Tento požadavek musí být splněn pro každý dotaz, který má databázový server zpracovat současně.
2. Nastavte parametr DBSPACETEMP na seznam nejméně dvou prostorů dbspace.

   Tento krok je vyžadován kvůli způsobu, jakým databázový server provádí vkládání. Při paralelním vkládání vytvoří databázový server nejprve fragmentovanou dočasnou tabulku. Aby databázový server věděl, kam má uložit fragmenty dočasné tabulky, musíte do konfiguračního parametru DBSPACETEMP nebo do proměnné prostředí **DBSPACETEMP** zadat seznam nejméně dvou prostorů dbspace. Konfigurační parametr DBSPACETEMP musíte také nastavit tak, aby před provedením příkazu SELECT...INTO určoval úložný prostor pro fragmenty.

Databázový server provádí paralelní vkládání zápisem do každého fragmentu současně metodou cyklické obsluhy. S vyšším počtem fragmentů roste i výkon.

Implicitní vkládání příkazem INSERT INTO...SELECT

Databázový server může také současně vkládat řádky do implicitních tabulek, které vytváří při zpracování příkazů SQL ve tvaru INSERT INTO...SELECT. Databázový server zpracovává současně například následující příkaz INSERT:

```
INSERT INTO target_table SELECT * FROM source_table
```

Cílová tabulka může být permanentní nebo dočasná.

Databázový server zpracovává tento typ příkazu INSERT současně pouze tehdy, splňuje-li cílová tabulka následující podmínky:

- Hodnota priority PDQ je větší než 0.
- Cílová tabulka je fragmentována nejméně do dvou prostorů dbspace.

- Cílová tabulka nemá povolena žádná referenční omezení ani spouštěče.
- Cílová tabulka není vzdálená tabulka.
- Cílová tabulka v databázi s protokolováním neobsahuje omezení filtrování.
- Cílová tabulka neobsahuje sloupce datových typů TEXT nebo BYTE.

Databázový server nezpracovává paralelní vkládání, která odkazují na rutinu SPL. Databázový server nezpracuje současně například následující příkaz:

```
INSERT INTO table1 EXECUTE PROCEDURE ins_proc
```

Paralelní vytváření indexů

Vytváření indexů může využívat výhod funkce PDQ a může je provádět současně. Databázový server provádí prohledávání i řazení současně s vytvářením indexů. Vytváření indexů lze vyvolat pomocí následujících operací:

- Vytvoření indexu.
- Přidání jedinečného primárního klíče.
- Přidání referenčního omezení.
- Povolení referenčního omezení.

Je-li v platnosti funkce PDQ, jsou prohledávání pro vytváření indexů řízena konfiguračním parametrem PDQ, který je popsán v části “Přidělování zdrojů paralelním databázovým dotazům” na stránce 12-7.

Máte-li v počítači několik procesorů, používá databázový server při řazení klíčů indexů dva jednotkové procesy. Během vytváření indexů bez uživatelem nastavené proměnné prostředí **PSORT_NPROCS** používá databázový server dva jednotkové procesy řazení.

Paralelní uživatelské rutiny

Obsahuje-li dotaz ve výrazu uživatelskou rutinu (UDR), může databázový server provést dotaz současně, pokud zapnete funkci PDQ. Databázový server může provádět následující paralelní operace, pokud je uživatelská rutina správně napsaná a zaregistrovaná:

- Paralelní prohledávání.
- Paralelní porovnávání s uživatelskou rutinou.

Další informace o tom, jak lze povolit paralelní provádění uživatelských rutin, naleznete v části “Paralelní uživatelské rutiny” na stránce 13-25.

Blokované kurzory, které používají funkci PDQ

Pokud blokované kurzory vytvořené deklarací kvalifikátoru WITH HOLD nemají žádné zámky, je funkce PDQ povolena. V následující případech bude funkce PDQ nastavena pro blokované kurzory:

- Dotazy s izolací na úrovni neaktualizovaného nebo potvrzeného čtení a kurzorem jen pro čtení.
- Dotazy s izolací na úrovni neaktualizovaného nebo potvrzeného čtení, které jsou jiné než ANSI a které nelze aktualizovat.

Operace databázového serveru, které nepoužívají funkci PDQ

Databázový server nezpracovává současně následující typy dotazů:

- Dotazy spuštěné s úrovní izolace stability kurzoru.

  Následné změny úrovně izolace nemají vliv paralelismus dotazů, které jsou již připraveny. Tato situace je důsledkem inherentní povahy paralelních prohledávání, která prohledávají několik řádků najednou.

- Dotazy, které používají kurzor deklarovaný jako FOR UPDATE.
- Příkaz UPDATE se spouštěčem *aktualizace*, který aktualizuje v části For Each Row definice spouštěče.
- Příkazy jazyka DDL (Data Definition Language)

  Úplný seznam naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Příkaz Update Statistics

Příkaz SQL UPDATE STATISTICS, který není zpracováván současně, je ovlivněn funkcí PDQ, protože musí přidělit paměť použitou při řazení. Chování příkazu UPDATE STATISTICS je ovlivněno správou paměti přidruženou k funkci PDQ.

I když příkaz UPDATE STATISTICS není zpracováván současně, musí databázový server přidělit paměť, kterou tento příkaz používá při řazení.

Rutiny SPL a spouštěče

Příkazy, které zahrnují rutiny SPL, nejsou prováděny současně. Ale příkazy v procedurách jsou prováděny současně.

Když databázový server provádí rutinu SPL, nepoužívá při zpracování nesouvisejících příkazů SQL obsažených v proceduře funkci PDQ. Každý příkaz SQL ale může být v případě potřeby prováděn současně pomocí paralelismu mezi dotazy. V důsledku toho byste měli omezit použití volání procedur z příkazů jazyka DML (Data Manipulation Language), chcete-li využívat možnosti paralelního zpracování databázového serveru. Úplný seznam příkazů jazyka DML naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Databázový server používá paralelismus mezi dotazy při zpracování příkazů ve spouštěči SQL stejným způsobem, jakým zpracovává příkazy v rutinách SPL.

Souvztažné a nesouvztažné poddotazy

Při zpracování souvztažných poddotazů nepoužívá databázový server funkci PDQ. Souvztažný poddotaz nemůže provádět současně více než jeden jednotkový proces. Zatímco jeden jednotkový proces provádí souvztažný poddotaz, jiné jednotkové procesy, které žádají o provedení souvztažného poddotazu, jsou blokovány dokud není první proces dokončen.

U nesouvztažných poddotazů provádí poddotaz pouze první jednotkový proces, který právě požádal o provedení. Ostatní jednotkové procesy pouze používají výsledky poddotazu a mohou je používat současně.

V důsledku toho důrazně doporučujeme, abyste při vytváření dotazů raději používali spojení namísto poddotazů, kdykoli je to možné. Dotazy pak budou moci využívat funkci PDQ.

Vnější spojení indexů

Databázový server snižuje prioritu PDQ dotazů, které obsahují vnější spojení indexů, na hodnotu LOW (pokud byla tato priorita nastavena na vyšší hodnotu) po dobu trvání tohoto dotazu. Obsahuje-li poddotaz nebo pohled vnější spojení indexů, sníží databázový server prioritu PDQ pouze tohoto poddotazu nebo pohledu, ale nesníží prioritu PDQ nadřazeného dotazu ani jiného poddotazu.

Vzdálené tabulky

I když databázový server může zpracovávat data uložená ve vzdálené tabulce současně, komunikace s těmito daty probíhá postupně, protože databázový server přiděluje jeden jednotkový proces jak pro předání, tak pro příjem dat ze vzdálené tabulky.

Databázový server sníží prioritu PDQ dotazů, které vyžadují přístup ke vzdálené databázi, na hodnotu LOW. V takovém případě jsou všechna místní prohledávání paralelní, ale místní spojení a vzdálený přístup nejsou paralelní.

Správce přidělující paměť

Správce přidělující paměť (MGM) je komponenta databázového serveru, která koordinuje použití paměti, virtuálních procesorů CPU, diskového vstupu - výstupu a jednotkových procesů prohledávání mezi dotazy pro podporu rozhodování. Pomocí konfiguračních parametrů DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS a MAX_PDQPRIORITY určuje správce MGM počet těchto zdrojů PDQ, které lze přidělit dotazu pro podporu rozhodování. Další informace o těchto konfiguračních parametrech uvádí Kapitola 4, “Vliv konfigurace na využití paměti”, na stránce 4-1.

Správce MGM dynamicky přiděluje dotazům pro podporu rozhodování následující zdroje:

- Počet jednotkových procesů prohledávání spuštěných pro každý dotaz pro podporu rozhodování
- Počet jednotkových procesů, které lze pro každý dotaz spustit
- Velikost paměti ve virtuální části sdílené paměti databázového serveru, kterou si může dotaz rezervovat

Používá-li databázový server intenzivně technologii OLTP a zjistíte, že se snižuje výkon, můžete pomocí prostředků správce MGM omezit zdroje určené dotazům pro podporu rozhodování. Mimo špičku můžete větší část zdrojů určit pro paralelní zpracování, a dosáhnout tak vyšší propustnosti dotazů pro podporu rozhodování.

Správce MGM přiděluje dotazu paměť pro takové aktivity, jako jsou například řazení, hashovací spojení a zpracování klauzulí GROUP BY. Velikost paměti, kterou používají dotazy pro podporu rozhodování, nesmí překročit hodnotu DS_TOTAL_MEMORY.

Správce MGM přiděluje dotazům paměť po *kvantech*. Kvantum můžete vypočítat podle následujícího vzorce:

```
kvantum paměti = DS_TOTAL_MEMORY / DS_MAX_QUERIES
```

Je-li například hodnota DS_TOTAL_MEMORY 12 megabajtů a hodnota DS_MAX_QUERIES je 4 MB, je kvantum 3 megabajty (12/4). Pro tyto platné hodnoty se tedy kvantum paměti rovná 3 MB. Obecně je paměť přidělována efektivněji, jsou-li kvanta menší. Často můžete zvýšit výkon souběžných dotazů tím, že zvýšíte hodnotu DS_MAX_QUERIES snížením velikosti kvanta paměti.

Chcete-li monitorovat zdroje, které přiděluje správce MGM, zadejte příkaz **onstat -g mgm** . Tento příkaz zobrazí pouze aktuálně používanou část paměti, nezobrazí celou přidělenou paměť. Další informace o tomto příkazu naleznete v části “Monitorování zdrojů správce MGM” na stránce 12-14.

Správce MGM také přiděluje maximální počet jednotkových procesů prohledávání na jeden dotaz na základě hodnot parametrů DS_MAX_SCANS a DS_MAX_QUERIES.

Následující vzorec udává maximální počet jednotkových procesů prohledávání na jeden dotaz:

```
scan_threads = min (nfrags, DS_MAX_SCANS * (pdqpriority / 100)
   * (MAX_PDQPRIORITY / 100))
```

nfrags počet fragmentů v tabulce s největším počtem fragmentů.

pdqpriority hodnota priority PDQ, která je nastavena buď proměnnou prostředí **PDQPRIORITY**, nebo příkazem SQL SET PDQPRIORITY.

Další informace o libovolném z těchto konfiguračních parametrů databázového serveru uvádí Kapitola 4, “Vliv konfigurace na využití paměti”, na stránce 4-1.

Proměnná prostředí **PDQPRIORITY** a příkaz SQL SET PDQPRIORITY požadují pro dotaz procentní část zdrojů PDQ. Pomocí konfiguračního parametru MAX_PDQPRIORITY můžete omezit procentní část požadovaných zdrojů, které může dotaz získat, a omezit dopad dotazů pro podporu rozhodování na zpracování technologie OLTP. Další informace naleznete v části “Omezení priority dotazů DSS” na stránce 12-7.

Přidělování zdrojů paralelním databázovým dotazům

Používá-li databázový server při současném provádění dotazu funkci PDQ, klade velké nároky na operační systém. Funkce PDQ využívá zejména tyto zdroje:

- Paměť
- virtuální procesory CPU
- Diskový vstup - výstup (pro fragmentované tabulky a dočasné prostory tabulek)
- Jednotkové procesy prohledávání

Při konfiguraci databázového serveru byste měli zvážit, jak použití funkce PDQ ovlivní uživatele technologie OLTP, aplikace pro podporu rozhodování a jiné aplikace.

Používání zdrojů databázovým serverem můžete řídit jedním z následujících způsobů:

- Omezením priority paralelních databázových dotazů.
- Úpravou velikosti paměti.
- Omezením počtu jednotkových procesů prohledávání.
- Omezením počtu souběžných dotazů.

Omezení priority dotazů DSS

Výchozí hodnotou priority PDQ jednotlivých aplikací je 0, to znamená, že zpracování paralelních databázových dotazů (PDQ) se nepoužívá. Databázový server tuto hodnotu používá, dokud ji nepotlačí jedna z následujících akcí:

- Uživatel nastavuje proměnnou prostředí **PDQPRIORITY**.
- Aplikace používá příkaz SET PDQPRIORITY.

Proměnná prostředí **PDQPRIORITY** a konfigurační parametr MAX_PDQPRIORITY spolupracují při řízení množství zdrojů přidělovaných pro paralelní zpracování. Správné nastavení těchto konfiguračních parametrů je velice důležité pro efektivní činnost funkce PDQ.

Konfigurační parametr MAX_PDQPRIORITY dovoluje administrátorovi databázového serveru omezit zdroje paralelního zpracování, které dotazy DSS využívají. Proměnná prostředí **PDQPRIORITY** tedy nastavuje *přiměřenou* nebo *doporučovanou* hodnotu priority a MAX_PDQPRIORITY omezuje zdroje, které může vyžadovat aplikace.

Konfigurační parametr MAX_PDQPRIORITY udává maximální procentní část požadovaných zdrojů, které může dotaz získat. Má-li například konfigurační parametr **PDQPRIORITY** hodnotu 80 a konfigurační parametr MAX_PDQPRIORITY má hodnotu 50, získá každý aktivní dotaz paměť o velikosti 40 % z hodnoty DS_TOTAL_MEMORY zaokrouhlenou dolů na nejbližší kvantum. V tomto příkladě omezuje konfigurační parametr

MAX_PDQPRIORITY efektivně počet souběžných dotazů pro podporu rozhodování na dva. Než mohou další dotazy získat zdroje potřebné ke spuštění, musejí čekat na skončení předchozích dotazů.

Aplikace nebo uživatel mohou použít hodnotu priority PDQ pomocí značky DEFAULT příkazu SET PDQPRIORITY, pokud byla tato hodnota nastavena proměnnou prostředí **PDQPRIORITY**. DEFAULT je symbolický ekvivalent hodnoty -1 pro prioritu PDQ.

Pomocí obslužného programu příkazového řádku **onmode** můžete dočasně změnit hodnoty následujících konfiguračních parametrů:

- Hodnotu konfiguračního parametru DS_TOTAL_MEMORY můžete změnit příkazem **onmode -M**.
- Hodnotu konfiguračního parametru DS_MAX_QUERIES můžete změnit příkazem **onmode -Q**.
- Hodnotu konfiguračního parametru MAX_PDQPRIORITY můžete změnit příkazem **onmode -D**.
- Hodnotu konfiguračního parametru DS_MAX_SCANS můžete změnit příkazem **onmode -S**.

Tyto změny zůstanou v platnosti pouze tak dlouho, dokud je databázový server zapnutý a spuštěný. Při spuštění používá databázový server hodnoty uvedené v souboru ONCONFIG.

Další informace o těchto konfiguračních parametrech uvádí Kapitola 4, “Vliv konfigurace na využití paměti”, na stránce 4-1. Další informace o obslužném programu **onmode** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Měníte-li hodnoty parametrů pro podporu rozhodování pravidelně (potřebujete-li například každý večer zpracovat sestavy a nastavujete konfigurační parametr MAX_PDQPRIORITY na hodnotu 100), můžete tyto hodnoty nastavovat pomocí plánované úlohy operačního systému. Další informace o vytváření plánovaných úloh naleznete v příručkách používaného operačního systému.

Chcete-li dosáhnout optimálního výkonu databázového serveru, zvolte hodnoty proměnné prostředí **PDQPRIORITY** a parametru MAX_PDQPRIORITY, pozorujte výsledné chování a potom upravte hodnoty těchto parametrů. Pro volbu hodnot této proměnné prostředí a tohoto parametru neexistují žádná vhodná pravidla. Následující části popisují strategie pro nastavení hodnot **PDQPRIORITY** a MAX_PDQPRIORITY pro specifické potřeby.

Omezení hodnoty priority PDQ

Konfigurační parametr MAX_PDQPRIORITY omezuje prioritu PDQ, kterou uděluje databázový server, když uživatel před vyvoláním dotazu buď nastaví proměnnou prostředí **PDQPRIORITY**, nebo zadá příkaz SET PDQPRIORITY. Při pokusu uživatele nebo aplikace o nastavení priority PDQ je udělená priorita vynásobena hodnotou, kterou udává konfigurační parametr MAX_PDQPRIORITY.

Nastavte nižší hodnotu konfiguračního parametru MAX_PDQPRIORITY, chcete-li přidělit více zdrojů zpracování technologie OLTP. Nastavte vyšší hodnotu, chcete-li přidělit více zdrojů zpracování podpory rozhodování. Rozsah možných hodnot je 0 až 100. Tento rozsah představuje procentní část zdrojů, které můžete přidělit ke zpracování podpory rozhodování.

Maximalizace propustnosti technologie OLTP

Někdy můžete chtít raději maximalizovat propustnost jednotlivých dotazů OLTP než propustnost dotazů pro podporu rozhodování. V takových případech nastavte konfigurační parametr MAX_ PDQPRIORITY na hodnotu 0. Omezíte prioritu PDQ na hodnotu OFF.

Hodnota OFF priority PDQ nebrání spuštění dotazů pro podporu rozhodování. Namísto toho způsobí, že budou dotazy spouštěny bez paralelizace. V této konfiguraci mohou být odezvy dotazů pro podporu rozhodování pomalé.

Hospodaření se zdroji

Jestliže aplikace dostatečně nevyužívají dotazy, které vyžadují paralelní řazení a paralelní spojení, zvažte nastavení priority PDQ na hodnotu LOW.

Pracuje-li databázový server v prostředí s více uživateli, mohli byste zvýšit výkon vzájemného dotazování za cenu paralelismu některých vnitřních dotazů nastavením konfiguračního parametru MAX_PDQPRIORITY na hodnotu 1. Mezi těmito dvěma různými typy paralelismu existuje dohoda, protože se ucházejí o stejné zdroje. Kompromisním řešením by mohlo být nastavení konfiguračního parametru MAX_PDQPRIORITY na nějakou střední hodnotu (možná 20 nebo 30) a nastavení proměnné prostředí **PDQPRIORITY** na hodnotu LOW. Proměnná prostředí nastavuje výchozí chování na hodnotu LOW, ale konfigurační parametr MAX_PDQPRIORITY umožňuje jednotlivým aplikacím, aby požadovaly více zdrojů pomocí příkazu SET PDQPRIORITY.

Umožnění maximálního využití paralelismu

Nastavte **PDQPRIORITY** a MAX_PDQPRIORITY na hodnotu 100, chcete-li, aby databázový server přiřadil paralelnímu zpracování tolik zdrojů, kolik potřebuje. Toto nastavení je vhodné v případě, že paralelní zpracování nekoliduje se zpracováním technologie OLTP.

Určení úrovně paralelismu

Pomocí různých číselných nastavení proměnné prostředí **PDQPRIORITY** můžete vyzkoušet vliv paralelismu na jednu aplikaci. Další informace o monitorování paralelního provádění naleznete v části “Monitorování zdrojů dotazů PDQ” na stránce 12-13.

Limity paralelismu přidruženého k prioritě PDQ

Databázový server snižuje prioritu PDQ dotazů, které obsahují vnější spojení, na hodnotu LOW (pokud byla nastavena na vyšší) po dobu provádění dotazu. Obsahuje-li poddotaz nebo pohled vnější spojení, sníží databázový server pouze prioritu PDQ tohoto dotazu nebo pohledu, ale nesníží prioritu nadřazeného dotazu ani jiného poddotazu.

Databázový server sníží prioritu PDQ dotazů, které vyžadují přístup ke vzdálené databázi (na stejné nebo odlišné instanci databázového serveru) na hodnotu LOW, pokud jste ji nastavili na vyšší hodnotu. V tomto případě jsou všechna místní prohledávání paralelní, ale místní spojení a vzdálené přístupy nejsou paralelní.

Použití rutin SPL

Databázový server zablokuje prioritu PDQ, která optimalizuje příkazy SQL v rutinách SPL při vytváření procedury nebo při poslední ruční rekompilaci příkazem UPDATE STATISTICS. Chcete-li změnit klientskou hodnotu **PDQPRIORITY**, vložte do rutiny SPL příkaz SET PDQPRIORITY.

Hodnota priority PDQ, kterou používá databázový server při optimalizaci nebo opětovné optimalizaci příkazu SQL, je hodnota nastavená příkazem SET PDQPRIORITY, který musel být proveden ve stejné proceduře. Pokud nebyl žádný takový příkaz proveden, používá se hodnota, která byla platná při poslední kompilaci nebo při posledním vytvoření procedury.

Hodnota priority PDQ, která je právě platná mimo proceduru, je při provádění procedury ignorována.

Doporučuje se vypnout prioritu PDQ při zadávání procedury a znovu ji zapnout pro specifické příkazy. Můžete zabránit blokování velkých objemů paměti touto procedurou a zajistit, aby nejdůležitější části procedury používaly vhodnou prioritu PDQ, jak ukazuje následující příklad:

```
CREATE PROCEDURE my_proc (a INT, b INT, c INT)
   Returning INT, INT, INT;
SET PDQPRIORITY 0;
...
SET PDQPRIORITY 85;
SELECT ... (rozsáhlý a složitý příkaz SELECT)
SET PDQPRIORITY 0;
...
;
```

Úprava velikosti paměti

Použijte následující vzorec jako počáteční bod pro odhad velikosti sdílené paměti, která má být přidělena dotazům pro podporu rozhodování:

```
DS_TOTAL_MEMORY = p_mem - os_mem - rsdnt_mem -
                  (128 kilobajtů * uživatelé) - other_mem
```

p_mem představuje celkovou fyzickou paměť, která je v hostitelském počítači dostupná.

os_mem představuje velikost operačního systému včetně mezipaměti.

resdnt_mem znamená velikost rezidentní sdílené paměti serveru Informix.

uživatelé znamená počet předpokládaných uživatelů (připojení) zadaný v konfiguračním parametru NETTYPE.

other_mem je velikost paměti používané ostatními aplikacemi (jinými než IBM Informix).

Hodnota konfiguračního parametru DS_TOTAL_MEMORY odvozená z tohoto vzorce slouží jen jako počáteční bod. Chcete-li zjistit vhodnou hodnotu pro svou konfiguraci, musíte , která monitorovat stránkování a odkládání. (Při monitorování stránkování a odkládání používejte nástroje dodávané s operačním systémem.) Zvýší-li se stránkování, sníží se hodnota parametru DS_TOTAL_MEMORY, takže zpracování zátěže OLTP může pokračovat.

Velikost paměti přidělované jednotlivému paralelnímu databázovému dotazu závisí na mnoha systémových faktorech, ale obecně je úměrná následujícímu vzorci:

```
memory_grant_basis = (DS_TOTAL_MEMORY/DS_MAX_QUERIES) *
               (PDQPRIORITY / 100) *
               (MAX_PDQPRIORITY / 100)
```

Pokud ale právě prováděné dotazy ve všech databázích dané instance serveru vyžadují více paměti, než kolik činí tento odhad průměrného přidělení, mohl by další dotaz způsobit přetečení disku nebo by mohl čekat, až souběžné dotazy dokončí provádění a uvolní dostatek paměťových zdrojů pro jeho provedení. Následující alternativní vzorec odhaduje paměť PDQ dostupnou pro jednotlivý dotaz přímo:

```
memory_for_single_query = DS_TOTAL_MEMORY  *
               (PDQPRIOIRTY / 100) *
               (MAX_PDQPRIORITY / 100)
```

Omezení počtu souběžných prohledávání

Aplikace databázového serveru přidělují dotazu určitý počet prohledávání podle jeho priority PDQ (kromě jiných faktorů). Konfigurační parametry DS_MAX_SCANS a MAX_PDQPRIORITY dovolují omezit zdroje, které mohou uživatelé dotazu přiřadit, podle následujícího vzorce:

```
scan_threads = min (nfrags, (DS_MAX_SCANS * (pdqpriority / 100)
   * (MAX_PDQPRIORITY / 100) )
```

nfrags počet fragmentů v tabulce s největším počtem fragmentů.

pdqpriority hodnota priority PDQ nastavená buď proměnnou prostředí **PDQPRIORITY**, nebo příkazem SET PDQPRIORITY.

Předpokládejme například, že rozsáhlá tabulka obsahuje 100 fragmentů. Bez povoleného omezení počtu souběžných prohledávání by čtení této tabulky vyžadovalo, aby databázový server souběžně prováděl 100 jednotkových procesů prohledávání. Navíc by tento dotaz mohl iniciovat libovolný počet uživatelů.

Nastavením konfiguračního parametru DS_MAX_SCANS na nižší hodnotu, než je počet fragmentů v této tabulce, můžete jako administrátor databázového serveru zabránit zaplavení databázového serveru jednotkovými procesy prohledávání prostřednictvím vícenásobných dotazů pro podporu rozhodování. Můžete nastavit konfigurační parametr DS_MAX_SCANS na hodnotu 20, a zajistit tak, aby databázový server prováděl souběžně nejvýše 20 jednotkových procesů prohledávání pro paralelní prohledávání. Pokud několik uživatelů iniciuje paralelní databázové dotazy, dostane každý dotaz pouze procentní část z 20 jednotkových procesů prohledávání podle priority PDQ přiřazené k dotazu a hodnoty MAX_PDQPRIORITY, kterou administrátor databázového serveru nastavil.

Omezení maximálního počtu dotazů

Konfigurační parametr DS_MAX_QUERIES omezuje počet souběžných dotazů pro podporu rozhodování, které lze spustit. Chcete-li odhadnout počet dotazů pro podporu rozhodování, které může databázový server spustit souběžně, počítejte každý dotaz, který je spuštěn s prioritou PDQ nastavenou na hodnotu 1 nebo vyšší, jako jeden úplný dotaz.

Databázový server přiděluje méně paměti dotazům, které běží s nižší prioritou, můžete tedy dotazům s nižší prioritou přiřadit hodnotu priority PDQ, která je mezi 1 a 30, v závislosti na vlivu dotazu na zdroje. Celkový počet dotazů s hodnotami priority PDQ většími než 0 nesmí překročit hodnotu konfiguračního parametru DS_MAX_QUERIES.

Správa aplikací

Administrátor databázového serveru, autor aplikace a uživatelé, ti všichni mají určitý podíl na řízení množství zdrojů, které databázový server přiděluje pro zpracování dotazu. Administrátor databázového serveru uplatňuje svůj podíl na řízení prostřednictvím konfiguračních parametrů. Vývojáři aplikací nebo uživatelé mohou svůj podíl na řízení uplatnit pomocí proměnné prostředí nebo příkazu SQL.

Použití příkazu SET EXPLAIN

Výstup příkazu SET EXPLAIN zobrazuje rozhodnutí, která provádí optimalizátor dotazů. Zobrazuje, zda byla použita paralelní prohledávání, maximální počet jednotkových procesů, které vyžaduje odpověď na daný dotaz a typ spojení použitého pro dotaz. Pomocí příkazu SET EXPLAIN můžete studovat plány dotazů pro aplikace. Dotaz můžete znovu sestavit nebo můžete použít proměnnou prostředí OPTCOMPIND, chcete-li změnit způsob, jakým optimalizátor s daným dotazem pracuje.

Použití proměnné prostředí a konfiguračního parametru OPTCOMPIND

Proměnná prostředí **OPTCOMPIND** a konfigurační parametr OPTCOMPIND označují preferovaný plán spojení, a pomáhají tak optimalizátoru vybrat vhodnou metodu spojení pro paralelní databázové dotazy.

Chcete-li ovlivnit volbu plánu spojení v optimalizátoru, můžete nastavit konfigurační parametr OPTCOMPIND. Hodnota, kterou přiřadíte tomuto konfiguračnímu parametru, je uplatněna pouze tehdy, pokud aplikace nenastaví proměnnou prostředí **OPTCOMPIND**.

Proměnnou prostředí OPTCOMPIND můžete nastavit na hodnotu 0, pokud chcete, aby databázový server vybral plán spojení přesně tak, jak ho vybíral ve verzích databázového serveru před verzí 6.0. Tato volba zajistí kompatibilitu s předchozími verzemi databázového serveru.

Aplikace s režimem izolace opakovatelné čtení může zamknout všechny záznamy v tabulce, provádí-li spojení hashovacích funkcí. Proto doporučujeme, abyste proměnnou prostředí OPTCOMPIND nastavili na hodnotu 1.

Chcete-li, aby optimalizátor provedl zjištění na základě nákladů bez ohledu na úroveň izolace aplikací, nastavte proměnnou prostředí OPTCOMPIND na hodnotu 2.

Poznámka: Hodnotu proměnné prostředí OPTCOMPIND v relaci můžete změnit příkazem SET ENVIRONMENT OPTCOMPIND. Další informace o použití tohoto příkazu naleznete v části “Použití příkazu ENVIRONMENT OPTCOMPIND pro nastavení hodnoty proměnné OPTCOMPIND v rámci relace” na stránce 3-10.

Další informace o proměnné prostředí OPTCOMPIND a různých plánech spojení naleznete v části “Plán dotazů” na stránce 10-2.

Použití příkazu SET PDQPRIORITY

Příkaz SET PDQPRIORITY umožňuje dynamické nastavení priority PDQ v aplikaci. Hodnotou priority PDQ může být libovolné celé číslo od -1 do 100.

Priorita PDQ nastavená příkazem SET PDQPRIORITY nahrazuje hodnotu proměnné prostředí **PDQPRIORITY**.

Značka DEFAULT pro příkaz SET PDQPRIORITY umožňuje aplikaci vrátit hodnotu priority PDQ na hodnotu nastavenou proměnnou prostředí, pokud byla nastavena. Další informace o příkazu SET PDQPRIORITY naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Uživatelské řízení zdrojů

Uživatel může označit prioritu PDQ dotazu buď nastavením proměnné prostředí **PDQPRIORITY**, nebo provedením příkazu SET PDQPRIORITY před vyvoláním dotazu. Priorita PDQ uživatelům umožňuje požadovat pro dotaz určitý objem zdrojů paralelního zpracování.

Mezi zdroji, které uživatel požaduje, a objemem zdrojů, které databázový server dotazu přidělí, mohou být rozdílné. Tento rozdíl je způsoben stanovením maximální hodnoty pro uživatelské zdroje, kterou nastavil administrátor databázového serveru pomocí konfiguračního parametru MAX_PDQPRIORITY a která je vysvětlena v následující části.

Řízení zdrojů administrátorem databázového serveru

Při správě celkového objemu zdrojů, které databázový server přiděluje paralelním databázovým dotazům, nastavuje administrátor databázového serveru proměnnou prostředí a konfigurační parametry, kterými se zabývá následující část.

Řízení zdrojů přidělených funkci PDQ

Zdroje přidělené funkci PDQ můžete řídit nastavením proměnné prostředí **PDQPRIORITY**. Dotazy, které před vyvoláním nemají nastavenou proměnnou prostředí **PDQPRIORITY**,

nepoužívají funkci PDQ. Chcete-li navíc zavést maximální hodnotu pro uživatelské úrovně priority PDQ, můžete nastavit konfigurační parametr MAX_PDQPRIORITY.

Nastavením proměnné prostředí **PDQPRIORITY** a konfiguračního parametru MAX_PDQPRIORITY uplatníte kontrolu nad zdroji, které databázový server přiděluje mezi aplikace OLTP a DSS. Je-li například zpracování technologie OLTP intenzivní zejména v určitém časovém období dne, měli byste nastavit konfigurační parametr MAX_PDQPRIORITY na hodnotu 0. Tento konfigurační parametr stanoví maximální hodnotu pro zdroje požadované uživateli, kteří používají proměnnou prostředí **PDQPRIORITY**. Funkce PDQ je tedy vypnuta, dokud znovu nenastavíte konfigurační parametr MAX_PDQPRIORITY na nenulovou hodnotu.

Řízení zdrojů přidělených dotazům pro podporu rozhodování

Řízení zdrojů, které databázový server přiděluje dotazům pro podporu rozhodování, můžete provádět pomocí konfiguračních parametrů DS_TOTAL_MEMORY, DS_MAX_SCANS a DS_MAX_QUERIES. Kromě nastavení limitů pro paměť pro podporu rozhodování a počet dotazů pro podporu rozhodování, které mohou být spuštěny souběžně, určuje databázový server pomocí těchto parametrů objem paměti, kterou má přidělit jednotlivým dotazům pro podporu rozhodování, jakmile je uživatelé předají. Databázový server k tomu potřebuje nejprve vypočítat jednotku paměti zvanou *kvantum* jako podíl hodnot konfiguračních parametrů DS_TOTAL_MEMORY a DS_MAX_QUERIES. Jakmile uživatel vyvolá dotaz, přidělí databázový server tolik procent dostupného kvanta, kolik je hodnota priority PDQ dotazu.

Počet souběžných prohledávání pro podporu rozhodování, které databázový server povoluje, můžete také omezit nastavením konfiguračního parametru DS_MAX_SCANS.

Předchozí verze databázového serveru umožňovaly nastavit konfigurační parametr priority PDQ v souboru ONCONFIG. Pokud daná aplikace závisí na globálním nastavení priority PDQ, můžete použít jednu z následujících metod:

Jen pro UNIX

- Definujte **PDQPRIORITY** jako sdílenou proměnnou prostředí v souboru informix.rc. Další informace o souboru informix.rc naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Konec Jen pro UNIX

Jen pro Windows

- Nastavte proměnnou prostředí **PDQPRIORITY** pro určitou skupinu prostřednictvím přihlašovacího profilu. Další informace o přihlašovacím profilu naleznete v příručce používaného operačního systému.

Konec Jen pro Windows

Monitorování zdrojů dotazů PDQ

Monitorujte zdroje (sdílenou paměť a jednotkové procesy), které správce MGM přidělil dotazům PDQ, a zdroje, které tyto dotazy právě používají.

Použití zdrojů PDQ můžete monitorovat následujícími způsoby:

- Spuštěním jednotlivých příkazů obslužného programu **onstat** shromáždíte informace o specifických aspektech probíhajícího dotazu.

- Chcete-li zapsat plán dotazů do výstupního souboru, před spuštěním dotazu proveďte příkaz SET EXPLAIN.

Použití obslužného programu onstat

Pomocí různých příkazů obslužného programu **onstat** můžete určit, kolik jednotkových procesů je aktivních, a sdílené paměťové zdroje, které tyto jednotkové procesy používají.

Monitorování zdrojů správce MGM

Chcete-li monitorovat způsob, jakým správce přidělující paměť koordinuje využití paměti a jednotkové procesy prohledávání, použijte volbu **onstat -g mgm**. Obslužný program **onstat** čte struktury sdílené paměti a poskytuje statistické údaje, které jsou přesné v okamžiku provádění dotazu.

Obrázek 12-1 na stránce 12-15 znázorňuje ukázkový výstup.

Výstup příkazu **onstat -g mgm** zobrazuje jednotku paměti zvanou *kvantum. Kvantum paměti* představuje jednotku paměti podle následující definice:

```
kvantum paměti = DS_TOTAL_MEMORY / DS_MAX_QUERIES
```

Následující vzorec představuje výpočet kvanta paměti pro hodnoty, které znázorňuje Obrázek 12-1 na stránce 12-15:

```
kvantum  paměti = 4000 kilobajtů / 5
                = 800 kilobajtů
```

Kvantum jednotkového procesu prohledávání je vždycky rovno 1.

```
Memory Grant Manager (MGM)
--------------------------

MAX_PDQPRIORITY:  100
DS_MAX_QUERIES:    5
DS_MAX_SCANS:      10
DS_TOTAL_MEMORY:   4000 KB

Queries:   Active     Ready   Maximum
               3         0         5

Memory:     Total      Free   Quantum
(KB)         4000      3872       800

Scans:      Total      Free   Quantum
              10         8         1

Load Control:    (Memory)      (Scans)  (Priority)  (Max Queries)   (Reinit)
                   Gate 1       Gate 2      Gate 3         Gate 4     Gate 5
(Queue Length)          0            0           0              0          0

Active Queries:
---------------
Session   Query  Priority  Thread   Memory  Scans      Gate
      7  a3d0c0      1     a8adcc    0/0       1/1       -
      7  a56eb0      1     ae6800    0/0       1/1       -
      9  a751d4      0     96b1b8   16/16      0/0       -

Ready Queries:  None

Free Resource        Average #        Minimum #
--------------    ---------------     ---------
Memory             489.2 +- 28.7          400
Scans                8.5 +- 0.5             8

Queries              Average #        Maximum #    Total #
---------------   ---------------     ---------    -------
Active               1.7 +- 0.7             3         23
Ready                0.0 +- 0.0             0          0

Resource/Lock Cycle Prevention count:  0
```

Obrázek 12-1. Výstup příkazu onstat -g mgm

Vysvětlení výstupu ukázkového příkazu onstat -g mgm: První část výstupu ukazuje hodnoty konfiguračních parametrů dotazu PDQ.

Druhá část výstupu popisuje informace o vnitřním řízení správce MGM. Obsahuje čtyři skupiny údajů.

První skupinou je skupina **Queries** (Dotazy). Tato skupina obsahuje následující informace:

Sloupec	**Popis**
Active	Počet dotazů PDQ, které jsou právě prováděny
Ready	Počet uživatelských dotazů připravených ke spuštění, jejichž provádění ale databázový server pozdržel z důvodů kontroly zatížení
Maximum	Maximální počet aktivních dotazů, které databázový server povoluje. Zobrazuje aktuální hodnotu konfiguračního parametru DS_MAX_QUERIES

Další skupinou je skupina **Memory** (Paměť). Tato skupina obsahuje následující informace:

Sloupec	**Popis**
Total	ERROR! SEGMENT DATA CORRUPTED, SEGDATA=

Sloupec	Popis
Free	Počet kilobajtů paměti pro dotazy PDQ, které právě nejsou používány
Quantum	Počet kilobajtů paměti v kvantu paměti

Další skupinou je skupina **Scans** (Prohledávání). Tato skupina obsahuje následující informace:

Sloupec	Popis
Total	Celkový počet jednotkových procesů prohledávání, který udává konfigurační parametr DS_MAX_SCANS
Free	Počet prohledávání, která jsou právě k dispozici pro dotazy pro podporu rozhodování
Quantum	Počet jednotkových procesů prohledávání v kvantu jednotkového procesu prohledávání

Poslední skupina v této části výstupu popisuje **kontrolu zatížení** správce MGM. Tato skupina obsahuje následující informace:

Sloupec	Popis
Memory	Počet dotazů, které čekají na paměť
Scans	Počet dotazů, které čekají na prohledávání
Priority	Počet dotazů, které čekají na dotazy s vyšší prioritou PDQ
Max Queries	Počet dotazů, které čekají na spuštění dotazu
Reinit	Počet dotazů, které čekají na dokončení spuštěných dotazů po příkazu **onmode -M**, **-p** nebo **-Q**

Další část výstupu, **Active Queries** (Aktivní dotazy) popisuje aktivní a připravené dotazy správce MGM. Tato část výstupu udává počet dotazů, které čekají u každé brány.

Sloupec	Popis
Session	ID relace, která daný dotaz vyvolala
Query	Adresa vnitřního kontrolního bloku přidruženého k danému dotazu
Priority	Priorita PDQ přiřazená k danému dotazu
Thread	Jednotkový proces, který registroval daný dotaz u správce MGM
Memory	Paměť aktuálně přidělená danému dotazu nebo paměť rezervovaná pro daný dotaz (jednotkou jsou stránky správce MGM, tj. 8 kilobajtů.)
Scans	Počet jednotkových procesů prohledávání aktuálně používaných daným dotazem nebo počet jednotkových procesů prohledávání přidělených danému dotazu
Gate	Číslo brány, u které daný dotaz čeká

Další část výstupu, **Free Resource** (Volné zdroje), poskytuje statistické údaje o volných zdrojích správce MGM. Čísla v této části a v poslední části představují statistické údaje od inicializace systému nebo od posledního příkazu **onmode -Q**, **-M** nebo **-S** . Tato část výstupu obsahuje následující informace:

Sloupec	Popis
Average	Průměrná velikost paměti a počet prohledávání
Minimum	Minimum dostupné paměti a počtu prohledávání

Poslední část výstupu, **Queries** (Dotazy), poskytuje statistické údaje týkající se dotazů správce MGM:

Sloupec	Popis
Average	Průměrná délka fronty aktivních a připravených dotazů
Maximum	Maximální délka fronty aktivních a připravených dotazů
Total	Celková délka fronty aktivních a připravených dotazů
Resource/Lock Cycle Prevention count	Udává, kolikrát systém provedl okamžitou aktivaci dotazu, aby zabránil možnému zablokování. Databázový server může zjistit, že některé dotazy mohou ve své frontě vytvořit situaci zablokování, pokud nebudou okamžitě spuštěny.

Monitorování jednotkových procesů dotazů PDQ

Informace o všech jednotkových procesech, které jsou spuštěny pro dotazy pro podporu rozhodování můžete získat pomocí voleb **onstat -u** a **onstat -g ath**.

Pomocí volby **onstat -u** můžete vypsat všechny jednotkové procesy pro danou relaci. Je-li v dané relaci spuštěn dotaz pro podporu rozhodování, obsahuje výstup primární jednotkový proces a všechny další jednotkové procesy. Obrázek 12-2 obsahuje například relaci 10, která má celkem pět jednotkových procesů.

```
Userthreads
address  flags   sessid   user     tty      wait     tout locks nreads   nwrites
80eb8c   ---P--D 0        informix -        0        0    0     33       19
80ef18   ---P--F 0        informix -        0        0    0     0        0
80f2a4   ---P--B 3        informix -        0        0    0     0        0
80f630   ---P--D 0        informix -        0        0    0     0        0
80fd48   ---P--- 45       chrisw   ttyp3    0        0    1     573      237
810460   ------- 10       chrisw   ttyp2    0        0    1     1        0
810b78   ---PR-- 42       chrisw   ttyp3    0        0    1     595      243
810f04   Y------ 10       chrisw   ttyp2    beacf8   0    1     1        0
811290   ---P--- 47       chrisw   ttyp3    0        0    2     585      235
81161c   ---PR-- 46       chrisw   ttyp3    0        0    1     571      239
8119a8   Y------ 10       chrisw   ttyp2    a8a944   0    1     1        0
81244c   ---P--- 43       chrisw   ttyp3    0        0    2     588      230
8127d8   ----R-- 10       chrisw   ttyp2    0        0    1     1        0
812b64   ---P--- 10       chrisw   ttyp2    0        0    1     20       0
812ef0   ---PR-- 44       chrisw   ttyp3    0        0    1     587      227
 15 active, 20 total, 17 maximum concurrent
```

Obrázek 12-2. Výstup příkazu onstat -u

Výstup příkazu **onstat -g ath** také obsahuje tyto jednotkové procesy a zahrnuje navíc sloupec **name**, který označuje roli jednotkového procesu. Jednotkové procesy, které spustil primární jednotkový proces podpory rozhodování, mají název, který označuje jejich roli v dotazu pro podporu rozhodování. Obrázek 12-3 znázorňuje čtyři jednotkové procesy *prohledávání* spuštěné primárním jednotkovým procesem **sqlexec**).

```
Threads:
tid     tcb     rstcb   prty    status                  vp-class   name
...
11      994060  0       4       sleeping(Forever)         1cpu    kaio
12      994394  80f2a4  2       sleeping(secs: 51)        1cpu    btclean
26      99b11c  80f630  4       ready                     1cpu    onmode_mon
32      a9a294  812b64  2       ready                     1cpu    sqlexec
113     b72a7c  810b78  2       ready                     1cpu    sqlexec
114     b86c8c  81244c  2       cond wait(netnorm)        1cpu    sqlexec
115     b98a7c  812ef0  2       cond wait(netnorm)        1cpu    sqlexec
116     bb4a24  80fd48  2       cond wait(netnorm)        1cpu    sqlexec
117     bc6a24  81161c  2       cond wait(netnorm)        1cpu    sqlexec
118     bd8a24  811290  2       ready                     1cpu    sqlexec
119     beae88  810f04  2       cond wait(await_MC1)      1cpu    scan_1.0
120     a8ab48  8127d8  2       ready                     1cpu    scan_2.0
121     a96850  810460  2       ready                     1cpu    scan_2.1
122     ab6f30  8119a8  2       running                   1cpu    scan_2.2
```

Obrázek 12-3. Výstup příkazu onstat -g ath

Monitorování zdrojů přidělených dané relaci

Pomocí volby **onstat -g ses** můžete monitorovat zdroje, které jsou přiděleny relaci, v níž je spouštěn dotaz pro podporu rozhodování, a které jsou touto relací používány. Volba **onstat -g ses** zobrazí následující informace:

- Sdílená paměť přidělená relaci, v níž je spouštěn dotaz pro podporu rozhodování
- Sdílená paměť používaná relací, v níž je spouštěn dotaz pro podporu rozhodování
- Počet jednotkových procesů, které databázový server spustil pro danou relaci

Obrázek 12-4 obsahuje například relaci s číslem 49, v níž je spuštěno pět jednotkových procesů pro dotaz pro podporu rozhodování.

```
session                                     #RSAM    total      used
id       user     tty      pid   hostname  threads  memory     memory
57       informix -        0     -         0        8192       5908
56       user_3   ttyp3    2318  host_10   1        65536      62404
55       user_3   ttyp3    2316  host_10   1        65536      62416
54       user_3   ttyp3    2320  host_10   1        65536      62416
53       user_3   ttyp3    2317  host_10   1        65536      62416
52       user_3   ttyp3    2319  host_10   1        65536      62416
51       user_3   ttyp3    2321  host_10   1        65536      62416
49       user_1   ttyp2    2308  host_10   5        188416     178936
2        informix -        0     -         0        8192       6780
1        informix -        0     -         0        8192       4796
```

Obrázek 12-4. Výstup příkazu onstat -g ses

Použití příkazu SET EXPLAIN

Je-li zapnuta funkce PDQ, ukazuje výstup příkazu SET EXPLAIN, zda optimalizátor zvolil paralelní prohledávání. Pokud optimalizátor zvolil paralelní prohledávání, zobrazí se ve výstupu slovo Parallel (Paralelní). Je-li funkce PDQ vypnuta, zobrazí se ve výstupu slovo Serial (Sériové).

Je-li zapnuta funkce PDQ, ukazuje optimalizátor maximální počet jednotkových procesů, od kterých je vyžadována odpověď na dotaz. Ve výstupu se zobrazí položka # of Secondary Threads (Počet sekundárních jednotkových procesů). Toto pole zobrazuje počet jednotkových procesů, od kterých je vyžadována odpověď, kromě jednotkového procesu vaší uživatelské relace. Celkový počet takto potřebných jednotkových procesů je počet sekundárních jednotkových procesů plus 1.

Následující příklad zobrazuje výstup příkazu SET EXPLAIN pro tabulku s fragmentací a prioritou PDQ nastavenou na hodnotu LOW:

```
SELECT * FROM t1 WHERE c1 > 20

Estimated Cost: 2
Estimated # of Rows Returned: 2

1) informix.t1: SEQUENTIAL SCAN (Parallel, fragments: 2)

 Filters: informix.t1.c1 > 20

# of Secondary Threads = 1
```

Následující příklad částečného výstupu příkazu SET EXPLAIN znázorňuje dotaz se spojením hashovací funkcí mezi dvěma fragmentovanými tabulkami a prioritou PDQ nastavenou na hodnotu ON. Tento dotaz je označen příkazem DYNAMIC HASH JOIN a tabulka, ve které je hashovací funkce vytvořena, je označena příkazem Build Outer.

```
QUERY:
------
SELECT h1.c1, h2.c1 FROM h1, h2 WHERE h1.c1 = h2.c1

Estimated Cost: 2
Estimated # of Rows Returned: 5

1) informix.h1: SEQUENTIAL SCAN (Parallel, fragments: ALL)

2) informix.h2: SEQUENTIAL SCAN (Parallel, fragments: ALL)


DYNAMIC HASH JOIN (Build Outer)
 Dynamic Hash Filters: informix.h1.c1 = informix.h2.c1

# of Secondary Threads = 6
```

Následující příklad částečného výstupu příkazu SET EXPLAIN znázorňuje tabulku s fragmentací, prioritou PDQ nastavenou na hodnotu LOW a indexem, který byl vybrán jako přístupový plán.

```
SELECT * FROM t1 WHERE c1 < 13

Estimated Cost: 2
Estimated # of Rows Returned: 1

1) informix.t1: INDEX PATH

 (1) Index Keys: c1 (Parallel, fragments: ALL)
 Upper Index Filter: informix.t1.c1 < 13


# of Secondary Threads = 3
```

Kapitola 13. Zvyšování výkonu jednotlivých dotazů

Obsah kapitoly

Tato kapitola obsahuje návrh způsobů použití obecných a koncepčních informací a informací o monitorování ke zvýšení výkonu dotazu.

Použití vyhrazeného testovacího systému

Pokud je to možné, můžete se rozhodnout testovat dotaz na systému, který nekoliduje s provozními databázovými servery. I když používáte databázový server jako datový sklad, je možné, že budete někdy testovat dotazy na odděleném systému, dokud neporozumíte problému ladění odpovídajícího dotazu. Testování dotazů na odděleném systému může různými způsoby zkreslit rozhodnutí při ladění.

Pokud se pokoušíte zvýšit výkon velkého dotazu, jehož dokončení může trvat několik minut nebo hodin, můžete připravit redukovanou databázi, ve které budou testy dokončeny rychleji. Berte však na vědomí následující možné problémy:

- Optimalizátor může v malé databázi provádět jiné volby než ve větší databázi, i když bude relativní velikost tabulek stejná. Ověřte, zda je ve skutečné databázi a modelové databázi stejný plán dotazů.
- Čas provedení je málokdy lineární funkcí velikosti tabulky. Například čas řazení se zvyšuje rychleji než velikost tabulky a nákladovost indexovaného přístupu přechází od dvou ke třem úrovním. To, co se zdá být velkým zlepšením v případě redukované databáze, může být bezvýznamné při použití na úplnou databázi.

Všechny závěry, které vytvoříte na základě výsledků testů v modelové databázi, je nutné považovat za prozatimní, dokud je neověříte na provozní databázi.

Zvýšení výkonu můžete často dosáhnout úpravou dotazu nebo datového modelu s tím, že budete mít na paměti následující cíle:

- Pokud používáte víceuživatelský systém nebo síť, kde se zatížení systému každou hodinu liší, pokuste se své experimenty provést každý den ve stejný čas, aby byly získány opakovatelné výsledky. Testování zahajte, když je zatížení systému trvale nízké, takže budete opravdu měřit pouze vliv svého dotazu.
- Pokud je dotaz vložen do složitého programu, můžete příkaz SELECT vyjmout a vložit jej do skriptu obslužného programu DB–Access.

Zobrazení plánu dotazů

Před změnou dotazu prostudujte plán tohoto dotazu a určete druh a množství zdrojů, které vyžaduje. Plán dotazů zobrazuje, jaká paralelní prohledávání jsou použita, maximální počet vyžadovaných jednotkových procesů, použité indexy atd. Poté vyzkoušením modelu dat ověřte, zda změny, které tato kapitola nevrhuje, tento model zlepší.

Plán dotazů můžete zobrazit některou z následujících metod:

- Před provedením dotazu proveďte jeden z následujících příkazů SET EXPLAIN:
  - SET EXPLAIN ON

    Tento příkaz jazyka SQL zobrazí zvolený plán dotazů. Popis výstupu příkazu SET EXPLAIN ON naleznete v části "Sestava, která zobrazuje plán dotazů zvolený optimalizátorem" na stránce 10-10.
  - SET EXPLAIN ON AVOID_EXECUTE

    Tento příkaz jazyka SQL zobrazí zvolený plán dotazů a neprovede žádný dotaz.
- Použijte v dotazu jednu z následujících direktiv EXPLAIN:
  - EXPLAIN
  - EXPLAIN AVOID_EXECUTE

Další informace o těchto direktivách EXPLAIN naleznete v části "Direktivy EXPLAIN" na stránce 11-10.

Zlepšení selektivity filtrů

S čím vyšší přesností určíte požadované řádky, tím vyšší bude pravděpodobnost, že dotaz bude dokončen rychle. Množství informací, které dotaz vyhodnocuje, můžete ovládat použitím klauzulí WHERE příkazu SELECT. Podmíněný výraz v klauzuli WHERE se obvykle nazývá *filtr*.

Informace o tom, jak selektivita filtrů ovlivňuje plán dotazů, které optimalizátor vybírá, naleznete v části "Filtry dotazu" na stránce 10-19. Následující části popisují některé pokyny, které vedou ke zlepšení selektivity filtrů.

Filtry s uživatelskými rutinami

Filtry dotazů mohou zahrnovat uživatelské rutiny (UDR). Selektivitu filtrů, které zahrnují uživatelské rutiny, můžete zlepšit následujícími vlastnostmi:

- Funkční indexy

  Funkční index můžete vytvořit ve výsledných hodnotách uživatelské rutiny nebo vestavěné funkce, která pracuje s jedním nebo více sloupci. Když vytvoříte funkční index, databázový server vypočítá vrácené hodnoty funkce a uloží je do indexu. Databázový server může vyhledat vrácené hodnoty funkce v odpovídajícím indexu bez provádění této funkce u každého vyhodnocovaného sloupce.

  Další informace o indexování uživatelských funkcí naleznete v části "Použití funkčního indexu" na stránce 7-18.
- Uživatelské funkce selektivity

  Můžete zapsat funkci, ketrá vypočítá očekávaný zlomek řádků, které jsou pro funkci vhodné. Stručný popis těchto uživatelských funkcí selektivity naleznete v části "Funkce selektivity a nákladovosti" na stránce 13-26. Další informace o tom, jak zapisovat a registrovat uživatelské funkce selektivity naleznete v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Vynechání některých filtrů

Pokud chcete dosáhnout nejlepšího výkonu, vyhněte se následujícím typům filtrů:

- Některé složité regulární výrazy
- Nepočáteční podřetězce

Následující části popisují tyto typy filtrů a příčiny, proč se jim vyhnout.

Vynechání složitých regulárních výrazů

Klíčová slova MATCHES a LIKE podporují shody typu *zástupný znak*, které jsou technicky známé jako *regulární výrazy*. Zpracování některých výrazů je pro databázový server obtížnější než zpracování výrazů ostatních. Zástupný znak v počáteční pozici, jak je uvedeno v následujícím příkladu (nalezení zákazníků, jejichž křestní jména nekončí na písmeno *y*), nutí databázový server ověřit každou hodnotu ve sloupci:

```
SELECT * FROM customer WHERE fname NOT LIKE '%y'
```

S tímto filtrem není možné použít index, takže k tabulce v tomto příkladu se musí přistupovat sekvenčně.

Pokud je složitý test regulárního výrazu nutný, vyhněte se mu kombinací se spojením. Pokud je to nutné, zpracujte jednu tabulku a použijte k výběru jednotlivých řádků test na regulární výrazy. Uložte výsledek do dočasné tabulky a spojte tuto tabulku s ostatními.

Testy na regulární výrazy se zástupnými znaky uprostřed nebo na konci operandu nezabrání použití indexu, pokud existuje.

Vynechání nepočátečních podřetězců

Filtr založený na nepočátečním podřetězci sloupce také vyžaduje, aby databázový server testoval každou hodnotu ve sloupci, jak uvádí následující příklad:

```
SELECT * FROM customer
   WHERE zipcode[4,5] > '50'
```

Databázový server nemůže k vyhodnocení takového filtru použít index.

Optimalizátor využívá index ke zpracování filtru, který testuje počáteční podřetězec indexovaného sloupce. Přítomnost testu podřetězce však může rušit použití kompozitního indexu k testování sloupce podřetězce a dalšího sloupce zároveň.

Použití filtrů spojení a filtrů po spojení

Databázový server poskytuje podporu části syntaxe spojení standardu ANSI, která zahrnuje následující klíčová slova:

- Klíčové slovo ON slouží k určení podmínky spojení a libovolných nepovinných filtrů spojení.
- Klíčová slova LEFT OUTER JOIN slouží k určení, která tabulka je dominantní (také se nazývá *vnější tabulka*)

Další informace o syntaxi spojení standardu ANSI naleznete v poznámkách k dokumentaci v příručce *IBM Informix Guide to SQL: Syntax*.

Ve vnějším spojení standardu ANSI provádí databázový server při zpracování filtru následující akce:

- Použije podmínku spojení v klauzuli ON k určení, které řádky podřízené tabulky (také se nazývá *vnitřní tabulka*) se mají spojit s vnější tabulkou.
- Před a během spojení použije v klauzuli ON nepovinné filtry spojení.

Pokud určíte filtr spojení v základní vnitřní tabulce v klauzuli ON, databázový server jej může použít před spojením, během prohledávání dat ve vnitřní tabulce. Filtry v základní podřízené tabulce v klauzuli ON mohou poskytnout následující další zlepšení výkonu:

- Méně řádků, které budou ve vnitřní tabulce prohledávány před spojením.
- Použití indexu k získání řádků z vnitřní tabulky před spojením.
- Méně spojovaných řádků.
- Méně řádků, které budou vyhodnocovány u filtrů v klauzuli WHERE.

Další informace o tom, co se stane, když určíte filtr spojení ve vnější tabulce v klauzuli ON, naleznete v poznámkách k dokumentaci v příručce *IBM Informix Guide to SQL: Syntax*.

- Po spojení použije filtry v klauzuli WHERE.

  Filtry v klauzuli WHERE mohou snížit počet řádků, které databázový server potřebuje prohledat, a počet řádků vracených uživateli.

  Termín *filtry po spojení* označuje filtry v klauzuli WHERE.

Když jsou prováděny distribuované dotazy, které k určení spojovaných tabulek a spojení vnořených smyček používají syntaxi LEFT OUTER kompatibilní se standardem ANSI, je všem zúčastněným databázovým serverům odeslán dotaz ke zpracování v místních tabulkách těchto serverů.

Ukázková databáze má tabulku **customer** a tabulku **cust_calls**, která zaznamenává hovory zákazníka s oddělením služeb. Předpokládejme, že určitý kód volání měl v minulosti mnoho výskytů a vy chcete zjistit, zda se hovory tohoto typu snížily. Pokud chcete zjistit, zda zákazníci již nemají tento kód volání, použijte na seznam všech zákazníků vnější spojení.

Obrázek Obrázek 13-1 zobrazuje ukázku příkazu jazyka SQL, která dokončí dotaz spojení kompatibilního se standardem ANSI a výstup příkazu SET EXPLAIN ON.

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
c.phone, u.call_dtime, u.call_code, u.call_descr
FROM customer c
LEFT JOIN cust_calls u ON c.customer_num = u.customer_num
ORDER BY u.call_dtime

Estimated Cost: 14
Estimated # of Rows Returned: 29
Temporary Files Required For: Order By

1) virginia.c: SEQUENTIAL SCAN

2) virginia.u: INDEX PATH

   (1) Index Keys: customer_num call_dtime (Serial, fragments: ALL)
     Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

  ON-Filters:virginia.c.customer_num = virginia.u.customer_num
  NESTED LOOP JOIN(LEFT OUTER JOIN)
```

Obrázek 13-1. Výstup příkazu SET EXPLAIN ON pro spojení standardu ANSI

Podívejte se na následující řádky ve výstupu příkazu SET EXPLAIN ON, které znázorňuje Obrázek 13-1:

- Řádek ON-Filters: vypisuje podmínku připojení, která byla určena v klauzuli ON.
- Poslední řádek výstupu příkazu SET EXPLAIN ON zobrazuje všechna tři klíčová slova (LEFT OUTER JOIN) pro spojení standardu ANSI, ačkoliv tento dotaz určuje v klauzuli FROM pouze klíčová slova LEFT JOIN. Klíčové slovo OUTER je nepovinné.

Obrázek 13-2 zobrazuje výstup příkazu SET EXPLAIN ON pro spojení standardu ANSI s filtrem spojení, který mezi voláními kontroluje kód volání I ve sloupci call_code.

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
c.phone, u.call_dtime, u.call_code, u.call_descr
FROM customer c LEFT JOIN cust_calls u
ON c.customer_num = u.customer_num
AND u.call_code = 'I'
ORDER BY u.call_dtime

Estimated Cost: 13
Estimated # of Rows Returned: 25
Temporary Files Required For: Order By

  1) virginia.c: SEQUENTIAL SCAN

  2) virginia.u: INDEX PATH

     Filters: virginia.u.call_code = 'I'

      (1) Index Keys: customer_num call_dtime   (Serial, fragments: ALL)
          Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

ON-Filters:(virginia.c.customer_num = virginia.u.customer_num
               AND virginia.u.call_code = 'I' )
NESTED LOOP JOIN(LEFT OUTER JOIN)
```

Obrázek 13-2. Výstup příkazu SET EXPLAIN ON pro filtr spojení ve spojení standardu ANSI

Hlavní rozdíly mezi výstupem, který znázorňuje Obrázek 13-1, a výstupem, který znázorňuje Obrázek 13-2, jsou následující:

- Optimalizátor volí k prohledávání vnitřní tabulky rozdílný index.

  Tento nový index využívá více filtrů a získá menší množství řádků. Spojení následně zpracovává méně řádků.
- Filtr spojení s klauzulí ON obsahuje další filtr.

Hodnota v řádku Estimated # of Rows Returned je pouze odhad a ne vždy odráží aktuální počet vrácených řádků. Ukázkový dotaz, který uvádí Obrázek 13-2, vrací díky dalšímu filtru méně řádků než dotaz, který uvádí Obrázek 13-1.

Obrázek 13-3 zobrazuje výstup příkazu SET EXPLAIN ON dotaz spojení standardu ANSI, který má v klauzuli WHERE filtr.

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
   c.phone, u.call_dtime, u.call_code, u.call_descr
FROM customer c LEFT JOIN cust_calls u
ON c.customer_num = u.customer_num
   AND u.call_code = 'I'
WHERE c.zipcode = "94040"
ORDER BY u.call_dtime

Estimated Cost: 3
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

  1) virginia.c: INDEX PATH

    (1) Index Keys: zipcode   (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.zipcode = '94040'

  2) virginia.u: INDEX PATH

     Filters: virginia.u.call_code = 'I'

    (1) Index Keys: customer_num call_dtime   (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

ON-Filters:(virginia.c.customer_num = virginia.u.customer_num
      AND virginia.u.call_code = 'I' )
NESTED LOOP JOIN(LEFT OUTER JOIN)

PostJoin-Filters:virginia.c.zipcode = '94040'
```

Obrázek 13-3. Výstup příkazu SET EXPLAIN ON pro filtr klauzule WHERE ve spojení standardu ANSI

Hlavní rozdíly mezi výstupem, který znázorňuje Obrázek 13-2 a výstupem, který znázorňuje Obrázek 13-3, jsou následující:

- Index ve sloupci **zipcode** ve filtru po spojení je zvolen pro dominantní tabulku.
- Řádek Filtry po spojení zobrazuje filtr v klauzuli WHERE.

Aktualizace statických údajů, nejsou-li generovány automaticky

Příkaz UPDATE STATISTICS aktualizuje v systémových katalozích statistické údaje, které optimalizátor využívá k určení plánu dotazů s nejnižší nákladovostí. Další údaje o konkrétních statistických údajích, které databázový server uchovává v tabulkách systémového katalogu, naleznete v části “Statistické údaje uchovávané pro tabulku a index” na stránce 10-18.

Důležité: Pokud jsou statistické údaje generovány automaticky, není nutné spouštět operace UPDATE STATISTICS.

Následující statistické údaje jsou automaticky generovány příkazem CREATE INDEX, ať s klíčovým slovem ONLINE, nebo bez něj:

- Statistické údaje na úrovni indexů, ekvivalentní statistickým údajům shromážděným operací UPDATE STATISTICS v režimu LOW, pro všechny typy indexů, včetně B-stromů, rozhraní Virtual Index Interface a funkčních indexů.
- Statistické údaje o distribuci sloupců, které jsou ekvivalentní s distribucí generovanou v operaci UPDATE STATISTICS v režimu HIGH, se vztahují na nekrycí hlavní indexovaný sloupec běžného indexu B-stromu.

Pokud chcete zajistit, aby optimalizátor vybral plán dotazů, který bude nejlépe odrážet aktuální stav tabulek, spouštějte příkaz UPDATE STATISTICS v pravidelných intervalech, pokud nejsou statistické údaje generovány automaticky.

Tip: Pokud předtím, než použijete obslužný program ON-BAR, spustíte příkaz UPDATE STATISTICS LOW v databázi **sysutils**, čas potřebný programem ON-BAR ke zpracování se zkrátí.

Následující tabulka shrnuje, kdy mají být spouštěny různé příkazy UPDATE STATISTICS, pokud nejsou statistické údaje generovány automaticky. Potřebujete-li spustit příkazy UPDATE STATISTICS a máte-li mnoho tabulek, můžete napsat skript a generovat příkazy UPDATE STATISTICS pomocí tohoto skriptu. Program ISA může generovat velké množství příkazů UPDATE STATISTICS pro tabulky.

Kdy provést	Příkaz UPDATE STATISTICS	Odkaz na podrobnosti a příklady	Program ISA generuje příkaz
Počet řádků se významně změnil nebo Po migraci z předchozí verze databázového serveru	UPDATE STATISTICS LOW DROP DISTRIBUTIONS	"Aktualizace počtu řádků" na stránce 13-8 nebo "Vypouštění distribucí dat" na stránce 13-9	Ne
Pro všechny sloupce, které nejsou hlavním sloupcem jakéhokoliv indexu	UPDATE STATISTICS LOW	"Vytvoření distribucí dat" na stránce 13-9	
Dotazy mají neindexované sloupce spojení nebo sloupce filtrů	UPDATE STATISTICS MEDIUM DISTRIBUTIONS ONLY	"Vytvoření distribucí dat" na stránce 13-9	Ano
Dotazy mají indexované sloupce spojení nebo sloupce filtrů	Tabulka UPDATE STATISTICS HIGH (hlavní sloupec indexu)	"Vytvoření distribucí dat" na stránce 13-9	Ano
Dotazy mají ve sloupcích spojení nebo sloupcích filtrů definován vícesloupcový index	Tabulka UPDATE STATISTICS HIGH (první odlišný sloupec ve vícesloupcovém indexu)	"Vytvoření distribucí dat" na stránce 13-9	Ne
Dotazy mají ve sloupcích spojení nebo sloupcích filtrů definován vícesloupcový index	Tabulka UPDATE STATISTICS LOW (všechny sloupce vícesloupcového indexu)	"Vytvoření distribucí dat" na stránce 13-9	Ne
Dotazy mají mnoho malých tabulek (vhodné do jedné oblasti)	UPDATE STATISTICS HIGH na malé tabulky	"Vytvoření distribucí dat" na stránce 13-9	Ne
Dotazy využívají rutiny SPL	UPDATE STATISTICS pro procedury	"Reoptimalizace rutin SPL" na stránce 10-30	Ne

Aktualizace počtu řádků

Při spuštění příkazu UPDATE STATISTICS LOW databázový server aktualizuje statistické údaje počtů tabulek, řádků a stránek v tabulkách systémového katalogu.

Spouštěním příkazu UPDATE STATISTICS LOW tak často, jak je to nutné, zajistíte, že statistické údaje pro počet řádků budou co nejaktuálnější. Pokud se kardinalita tabulky často mění, spouštějte pro tuto tabulku příkaz častěji.

Režim LOW je výchozí režim příkazu UPDATE STATISTICS. Následující ukázka příkazu jazyka SQL aktualizuje statistické údaje tabulek systémového katalogu **systables**, **syscolumns** a **sysindexes**, ale neaktualizuje distribuce dat:

```
UPDATE STATISTICS FOR TABLE tab1;
```

Vypouštění distribucí dat

Je možné, že při inovaci na novou verzi databázového serveru bude nutné vypustit distribuce a odstranit tak starou strukturu distribucí v tabulce systémového katalogu **sysdistrib**.

```
UPDATE STATISTICS DROP DISTRIBUTIONS;
```

Vytvoření distribucí dat

Při každém provedení příkazu UPDATE STATISTICS MEDIUM nebo UPDATE STATISTICS HIGH vytvoří databázový server distribuce dat, které poskytnou informace optimalizátoru. (V případě, že jsou statistické údaje generovány automaticky, není nutné operace UPDATE STATISTICS spouštět.)

Důležité:

Databázový server vytvoří distribuce dat vzorkováním dat sloupců, řazením dat, vytvořením distribučních zásobníků a vložením výsledků do tabulky systémového katalogu **sysdistrib**.

Velikost vzorků použitých při prohledávání můžete řídit klíčovými slovy HIGH nebo MEDIUM. Rozdíl nezi příkazy UPDATE STATISTICS HIGH a UPDATE STATISTICS MEDIUM je v počtu vzorkovaných řádků. Příkaz UPDATE STATISTICS HIGH vzorkuje celou tabulku, zatímco příkaz UPDATE STATISTICS MEDIUM vzorkuje pouze část řádků, založenou na hodnotě jistoty a rozlišení použitých v příkazu UPDATE STATISTICS.

Klíčové slovo LOW je možné použít spolu s příkazem UPDATE STATISTICS pouze pro zcela vhodné klíče indexů.

Pokud byla pro sloupec vytvořena distribuce, optimalizátor využije tuto informaci k odhadnutí počtu řádků, které odpovídají dotazu na sloupec. Když optimalizátor odhadne počet vrácených řádků, distribuce dat v tabulce **sysdistrib** nahradí hodnoty sloupců **colmin** a **colmax** tabulky systémového katalogu **syscolumns**.

Při prvním použití statistických údajů distribucí dat zkuste ve všech tabulkách aktualizovat statistické údaje v režimu MEDIUM a poté aktualizujte v režimu HIGH statistické údaje všech sloupců s indexy. Tato strategie vytvoří pro určené sloupce odhady statistických dotazů. Tyto odhady mají v průměru odchylku menší než *procento* z celkového počtu řádků v tabulce, kde *procento* je hodnota, kterou určíte v klauzuli RESOLUTION režimu MEDIUM. Výchozí hodnota pro režim MEDIUM je 2,5 procent. (Pro sloupce s distribucemi v režimu HIGH je výchozí rozlišení 0,5 procenta.)

Pomocí volby DISTRIBUTIONS ONLY můžete provést příkaz UPDATE STATISTICS MEDIUM na úrovni tabulky nebo celého systému, protože zahlcení díky sloupcům navíc není velké.

Databázový server používá paměťová místa určená proměnnou prostředí DBSPACETEMP pouze tehdy, používáte-li volbu HIGH příkazu UPDATE STATISTICS.

Nastavíte-li třetí parametr proměnné prostředí DBUPSPACE na hodnotu 1, můžete operacím UPDATE STATISTICS zabránit v použití indexů při řazení řádků.

Pro každou tabulku, ke které dotaz přistupuje, vytvořte distribuce dat podle následujících pokynů. Také si všimněte příkladů pod pokyny.

Postup vytvoření statistických údajů tabulky:

1. Identifikujte sadu všech sloupců, které se zobrazují ve všech jednosloupcových či vícesloupcových indexech tabulky.

2. Identifikujte část obsahující všechny sloupce, které nejsou hlavními sloupci žádného indexu.
3. Spusťte pro každý sloupec v této části příkaz UPDATE STATISTICS LOW.

Postup při vytvoření distribucí dat pro každou tabulku, ke které dotaz přistupuje:

1. Spusťte jeden příkaz UPDATE STATISTICS MEDIUM pro všechny sloupce v tabulce, které nemají index.

   Pokud tabulka není příliš velká, použijte výchozí parametry, jinak byste měli použít rozlišení 1,0 a jistotu 0,99.
2. Spuštěním následujícího příkazu UPDATE STATISTICS vytvoříte distribuce pouze pro neindexované sloupce spojení a neindexované sloupce filtrů:

   ```
   UPDATE STATISTICS MEDIUM DISTRIBUTIONS ONLY;
   ```
3. Pro všechny sloupce, které mají index, spusťte příkaz UPDATE STATISTICS HIGH. Pokud chcete zajistit nejrychlejší provádění příkazu UPDATE STATISTICS, je nutné příkaz UPDATE STATISTICS HIGH provést u každého sloupce, jak je zobrazeno v příkladu pod touto procedurou.
4. Pokud máte indexy, které začínají se stejnou částí sloupců, spusťte příkaz UPDATE STATISTICS HIGH u prvního sloupce každého indexu, který se liší, jak je zobrazeno v příkladu pod touto procedurou.
5. U každého jednosloupcového nebo vícesloupcového indexu tabulky:
   a. Identifikujte sadu všech sloupců, které se v indexu zobrazují.
   b. Identifikujte část obsahující všechny sloupce, které nejsou hlavními sloupci žádného indexu.
   c. Spusťte pro každý sloupec v této části příkaz UPDATE STATISTICS LOW. (Volba LOW je výchozí.)
6. U tabulek, ve kterých byly v kroku 3 vytvořeny indexy, aktualizujte tabulky systémového katalogu **sysindexes** a **syscolumns** spuštěním příkazu UPDATE STATISTICS podle následujícího příkladu:

   ```
   UPDATE STATISTICS FOR TABLE t1(a,b,e,f);
   ```
7. U malých tabulek spusťte příkaz UPDATE STATISTICS HIGH, například:

   ```
   UPDATE STATISTICS HIGH FOR TABLE t2;
   ```

Protože tento příkaz vytvoří statistické údaje pro každý index pouze jednou, tyto kroky zajistí, že příkaz UPDATE STATISTICS bude prováděn rychle.

Příklady příkazů UPDATE STATISTICS HIGH pro všechny sloupce, které mají indexy:

Předpokládejte, že máte tabulku **t1** se sloupci **a**, **b**, **c**, **d**, **e** a **f** s následujícími indexy:

```
CREATE INDEX ix_1 ON t1 (a, b, c, d) ... CREATE INDEX ix_3 ON t1 (f) ...
```

U sloupců, které mají indexy, spusťte následující příkazy UPDATE STATISTICS:

```
UPDATE STATISTICS HIGH FOR TABLE t1(a);
UPDATE STATISTICS HIGH FOR TABLE t1(f);
```

Tyto příkazy UPDATE STATISTICS HIGH nahradí u sloupců s indexy distribuce vytvořené příkazy UPDATE STATISTICS MEDIUM s distribucemi typu high.

Příklad příkazů UPDATE STATISTICS HIGH pro první sloupec v každém indexu, který se liší:

Předpokládejte, že máte v tabulce **t1** například následující indexy:

```
CREATE INDEX ix_1 ON t1 (a, b, c, d) ...
CREATE INDEX ix_2 ON t1 (a, b, e, f) ...
CREATE INDEX ix_3 ON t1 (f) ...
```

Krok 3 provede příkaz UPDATE STATISTICS HIGH ve sloupci **a** a ve sloupci **f**. Poté spusťte příkaz UPDATE STATISTICS HIGH ve sloupcích **c** a **e**.

```
UPDATE STATISTICS HIGH FOR TABLE t1(c);
UPDATE STATISTICS HIGH FOR TABLE t1(e);
```

Kromě toho můžete příkaz UPDATE STATISTICS HIGH spustit ve sloupci **b**, ačkoliv to obvykle není nutné.

Předpokládejme, že máte indexy, které začínají stejnou částí sloupců, ale obsahují některé sloupce, které se liší. Další informace o distribucích dat a příkazu UPDATE STATISTICS naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Aktualizace statistických údajů pro sloupce spojení

Díky zlepšením a upraveným odhadům nákladovosti při vytváření lepších plánů dotazů závisí optimalizátor v určitých případech velkou části na přesném porozumění základním informacím distribucí dat. Možná si stále myslíte, že složitý dotaz nebude proveden dostatečně rychle, ačkoliv jste se řídili pokyny v části "Vytvoření distribucí dat" na stránce 13-9. Pokud váš dotaz zahrnuje předpoklady rovnosti, proveďte jednu z následujících akcí:

- U sloupců spojení, které se zobrazují v klauzuli WHERE dotazu, spusťte příkaz UPDATE STATISTICS s klíčovým slovem HIGH. Pokud jste se řídili pokyny v části "Vytvoření distribucí dat" na stránce 13-9, sloupce, které mají indexy, již budou mít distribuce v režimu HIGH.
- Pokud chcete určit, zda informace o distribucích v režimu HIGH u sloupců, které nemají indexy, může při provádění poskytnout lepší cestu, řiďte se následujícími kroky:

Postup určení, zda může mít příkaz UPDATE STATISTICS HIGH použitý na sloupce spojení odlišné výsledky:

1. Proveďte příkaz SET EXPLAIN ON a opětovně spusťte dotaz.
2. Všimněte si odhadovaného počtu řádků ve výstupu příkazu SET EXPLAIN a skutečného počtu řádků, které dotaz vrátí.
3. Pokud se tato dvě čísla výrazně liší, spusťte příkaz UPDATE STATISTICS HIGH u sloupců, které se zapojují do spojení, pokud jste tak již neučinili.

Důležité: Pokud je tabulka příliš velká, provedení příkazu UPDATE STATISTICS s režimem HIGH může trvat dlouho.

Následující příklad uvádí dotaz, který obsahuje sloupce spojení:

```
SELECT employee.name, address.city
   FROM employee, address
   WHERE employee.ssn = address.ssn
   AND employee.name = 'James'
```

V tomto příkladu jsou sloupce spojení pole **ssn** v tabulkách **employee** a **address**. Distribuce dat pro oba z těchto sloupců musí přesně odpovídat aktuálním datům, aby optimalizátor mohl přesně určit plán spojení a pořadí provedení.

Příkaz UPDATE STATISTICS není možné použít k vytvoření distribucí dat pro tabulku, která je vůči aktuální databázi externí. Další informace o distribucích dat a příkazu UPDATE STATISTICS naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Aktualizace statistických údajů sloupců s uživatelskými datovými typy

Protože databázový server nemá k dispozici informace o vlastnostech a použití uživatelských datových typů (UDT), nemůže u uživatelských datových typů shromažďovat sloupce **colmin** a **colmax** tabulky systémového katalogu **syscolumns**. Pokud je nutné shromažďovat statistické údaje sloupců s uživatelskými datovými typy, musí programátoři napsat funkce, které rozšíří příkaz UPDATE STATISTICS. Další informace naleznete v kapitole věnované výkonu v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Protože distribuce dat uživatelských datových typů mohou být velké, je možné je volitelně uložit v prostoru sbspace namísto v tabulce systémového katalogu **sysdistrib**.

Postup uložení distribucí dat uživatelských datových typů do prostoru sbspace:

1. Pomocí příkazu **onspaces -c -S** vytvořte prostor sbspace.

   Pokud chcete zajistit obnovitelnost distribucí dat, určete ve volbě **-Df** možnost LOGGING=ON, jak uvádí následující příklad:

   ```
   % onspaces -c -S distrib_sbsp -p /dev/raw_dev1 -o 500 -s
         20000
      -m /dev/raw_dev2 500 -Ms 150 -Mo 200 -Df
         "AVG_LO_SIZE=32,LOGGING=ON"
   ```

   Informace o změně velikosti prostoru sbspace naleznete v části “Odhad stránek, které zabírají inteligentní velké objekty” na stránce 6-11.

   Další informace o určení charakteristiky úložiště prostoru sbspaces naleznete v části “Konfigurační parametry, které ovlivňují vstup - výstup prostoru sbspace” na stránce 5-18.
2. V konfiguračním parametru SYSSBSPACENAME určete prostor sbspace, který jste vytvořili v kroku 1.
3. Při spuštění příkazu UPDATE STATISTICS s klíčovým slovem MEDIUM nebo HIGH ke generování distribucí dat určete sloupec s uživatelským datovým typem.

Pokud chcete vytisknout distribuce dat pro sloupec s uživatelským datovým typem, použijte volbu **dbschema -hd**.

Použití aktualizace statistických údajů na velmi velké databáze

Pokud máte extrémně velkou databázi a tabulky a indexy jsou fragmentované, můžete spustit příkazy UPDATE STATISTICS paralelně. K provedení této akce je nutné spustit příkaz UPDATE STATISTICS LOW s prioritou PDQ nastavenou na hodnotu, která se nachází mezi hodnotami 1 a 10. Pokud je priorita PDQ nastavena na hodnotu 1, bude u aktuální tabulky analyzováno najednou 10 % fragmentů indexů. Pokud je priorita PDQ nastavena na hodnotu 5, bude u aktuální tabulky analyzováno najednou 50 % fragmentů indexů. Pokud je priorita PDQ nastavena na hodnotu 10, budou u aktuální tabulky analyzovány všechny fragmenty najednou.

Zobrazení distribucí

Pokud se hodnoty sloupců příliš nemění, není nutné regenerovat distribuce dat. Přesnost distribuce ověříte porovnáním výstupu příkazu **dbschema -hd** s výsledky vhodně sestavených příkazů SELECT.

Například následující příkaz **dbschema** vytvoří seznam distribucí pro každý sloupec tabulky **customer** databáze **vjp_stores** s počtem hodnot v každém zásobníku a počtem odlišných hodnot:

```
dbschema -hd customer -d vjp_stores
```

Obrázek 13-4 zobrazuje distibuce dat pro sloupec **zipcode**, který tento příkaz **dbschema -hd** vytvoří. Protože tento sloupec má index **zip_ix**, byl u něj spuštěn příkaz UPDATE STATISTICS HIGH, jak ukazuje výstupní řádek:

```
High Mode, 0.500000 Resolution
```

Obrázek 13-4 zobrazuje 17 zásobníků s jednou odlišnou hodnotou **zipcode** v každém zásobníku.

```
dbschema -hd customer -d vjp_stores

...
Distribution for virginia.customer.zipcode

Constructed on 09/18/2000

High Mode, 0.500000 Resolution

--- DISTRIBUTION ---

     (          02135 )
  1: (  1,   1, 02135 )
  2: (  1,   1, 08002 )
  3: (  1,   1, 08540 )
  4: (  1,   1, 19898 )
  5: (  1,   1, 32256 )
  6: (  1,   1, 60406 )
  7: (  1,   1, 74006 )
  8: (  1,   1, 80219 )
  9: (  1,   1, 85008 )
 10: (  1,   1, 85016 )
 11: (  1,   1, 94026 )
 12: (  1,   1, 94040 )
 13: (  1,   1, 94085 )
 14: (  1,   1, 94117 )
 15: (  1,   1, 94303 )
 16: (  1,   1, 94304 )
 17: (  1,   1, 94609 )


--- OVERFLOW ---

  1: (  2,      94022 )
  2: (  2,      94025 )
  3: (  2,      94062 )
  4: (  3,      94063 )
  5: (  2,      94086 )
```

Obrázek 13-4. Zobrazení distribucí dat pomocí příkazu dbschema -hd

Část výstupu OVERFLOW zobrazuje duplicitní hodnoty, které mohou distribuce dat posunout, takže je program **dbschema** přesune z distribuce na oddělený seznam. Počet duplikátů v tomto seznamu přetečení musí být větší než kritické množství, které určuje následující vzorec. Obrázek 13-4 zobrazuje hodnotu rozlišení .0050. Tento vzorec tedy určujte, že každá hodnota, která je duplikovaná více než jednou, je vypsána do části přebytku.

```
Přebytek = ,25 * rozlišení * počet_řádků
      = ,25 * ,0050 * 28
      = ,035
```

Další informace o obslužném programu **dbschema** naleznete v příručce*IBM Informix Migration Guide*.

Zvýšení výkonu příkazu UPDATE STATISTICS

Při provádění příkazu UPDATE STATISTICS používá databázový server k řazení a vytváření distribucí dat paměť a disk. Množství dostupné kapacity paměti a disku pro příkaz UPDATE STATISTICS můžete ovlivnit následujícími metodami:

- priorita PDQ

  I když příkaz UPDATE STATISTICS není zpracováván současně, můžete získat více paměti pro řazení nastavením priority PDQ na hodnotu větší než 0. Výchozí hodnotou priority PDQ je 0. Prioritu PDQ můžete nastavit buď pomocí proměnné prostředí **PDQPRIORITY**, nebo pomocí příkazu SET PDQPRIORITY jazyka SQL.

  Další informace o prioritě PDQ naleznete v části "Přidělování zdrojů paralelním databázovým dotazům" na stránce 12-7.

- proměnná prostředí **DBUPSPACE**

  Proměnnou prostředí **DBUPSPACE** můžete použít k omezení množství místa na systémovém disku, které může příkaz UPDATE STATISTICS použít k souběžnému vytváření vícesloupcových distribucí.

  Další informace o této proměnné prostředí naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Zvýšení výkonu pomocí indexů

Výkon dotazu můžete často zvýšit přidání nebo v některých případech odebrání indexů. Pokud chcete zvýšit výkon dotazu, zvažte použití některých metod, které popisuje následující část.

Kromě toho zvažte použití příkazů CREATE INDEX ONLINE a DROP INDEX ONLINE k vytvoření a vypuštění indexu v prostředí online, když jsou databáze a přidružené tabulky trvale dostupné. Tyto příkazy jazyka SQL umožňují vytvářet a vypouštět indexy, aniž by bylo nutné uzamknout tabulku po dobu vytváření nebo vypouštění indexu. Další informace naleznete v části "Vytvoření a vypuštění indexu v online prostředí" na stránce 7-10.

Nahrazení automatických indexů trvalými indexy

Pokud plán dotazů obsahuje cestu *automatického indexu* k velké tabulce, berte to jako doporučení optimalizátoru, které sděluje, že je možné zvýšit výkon přidáním indexu k tomuto sloupci. Pokud provádíte dotaz příležitostně, můžete server nechat v rozumné míře vytvořit a vyřadit index. Pokud provádíte dotaz pravidelně, ušetříte čas vytvořením trvalého indexu.

Použití složených indexů

Optimalizátor může použít složený index (index, který pokrývá více než jeden sloupec) různými způsoby. Databázový server může použít index na sloupce **a**, **b** a **c** (v tomto pořadí) následujícími způsoby:

- K vyhledání určitého řádku

  Databázový server může použít složený index, pokud je první filtr rovnosti a následující sloupce mají výrazy rozsahu (<, <=, >, >=). Následující příklady filtrů používají sloupce ve složeném indexu:

  ```
  WHERE a=1
  WHERE a>=12 AND a<15
  WHERE a=1 AND b < 5
  WHERE a=1 AND b = 17 AND c >= 40
  ```

  Následující příklady filtrů nemohou používat složený index:

  ```
  WHERE b=10
  WHERE c=221
  WHERE a>=12 AND b=15
  ```

- K nahrazení výsledku prohledávání tabulky, pokud jsou všechny požadované sloupce obsaženy v indexu

  Prohledávání, které využívá index, ale neodkazuje na tabulku, se nazývá *vyhledávání pouze klíčů*.
- Ke spojení sloupce **a**, sloupců **ab** nebo sloupců **abc** s jinou tabulkou
- K implementaci klauzule ORDER BY nebo GROUP BY na sloupce **a**, **ab** nebo **abc**, ale ne na sloupce **b**, **c**, **ac** nebo **bc**

Provedení je nejefektivnější, pokud vytvoříte složený index se sloupci v pořadí od nejvíce k nejméně odlišnému. Jinými slovy, sloupec, který vrátí největší počet odlišných řádků, pokud je v příkazu SELECT dotázán klíčovým slovem DISTINCT, by měl být ve složeném indexu první.

Pokud aplikace provádí několik dlouhých dotazů, z nichž každý obsahuje klauzule ORDER BY nebo GROUP BY, můžete někdy zlepšit výkon přidáním indexů, které vytvoří toto pořadí, aniž by vyžadovaly seřazení. Například následující dotaz seřadí každý sloupec v klauzuli ORDER BY v různém směru:

```
SELECT * FROM t1 ORDER BY a, b DESC;
```

Pokud se chcete vyhnout dočasným tabulkám při řazení sloupce **a** v vzestupném pořadí a sloupce **b** v sestupném pořadí, je nutné vytvořit složený index na sloupcích (**a**, **b** DESC) nebo (**a** DESC, **b**). Díky obousměrnému procházení stačí vytvořit pouze jeden z těchto indexů. Další informace o obousměrném procházení naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Provedení prohledávání tabulky a řazení výsledků namísto použití složeného indexu může být na druhou stranu méně nákladné, pokud jsou splněny tyto podmínky:

- Tabulka je vzhledem k indexu dobře seřazená.
- Počet řádků, které se z tabulky získají, představuje velkou procentní část dostupných dat.

Použití indexů v aplikacích datových skladů

Mnoho datových skladů využívá *schéma typu hvězda*. Schéma typu hvězda se skládá z tabulky *faktů* a několika *rozměrových* tabulek. Tabulka faktů je obvykle velká a obsahuje kvantitativní nebo faktické informace o předmětu. Rozměrová tabulka popisuje atribut tabulky faktů.

Pokud rozměr potřebuje informace nižší úrovně, je rozměr modelován pomocí hierarchie tabulek, která se nazývá *schéma typu sněhová vločka*.

Další informace o schématech typu hvězda a sněhová vločka naleznete v příručce *IBM Informix Database Design and Implementation Guide*.

Dotazy, které používají tabulky ve schématu hvězda nebo sněhová vločka, mohou využívat odpovídající index tabulky faktů. Vezměte v úvahu příklad schématu typu hvězda s jednou tabulkou faktů nazvanou **orders** a čtyřmi rozměrovými tabulkami nazvanými **customers**, **dodavatelé**, **produkty** a **clerks**. Tabulka **orders** popisuje podrobnosti jednotlivých objednávek, které obsahují identifikátor zákazníka, identifikátor dodavatele, identifikátor produktu a identifikátor prodavače. Každá rozměrová tabulka podrobně popisuje indentifikátor. Tabulka **orders** je velká a čtyři rozměrové tabulky jsou malé.

Následující dotaz zjistí celkové přímé tržby v oblasti Menlo Park (poštovní směrovací číslo 94025) pro pevné disky dodané dodavatelem Johnson:

```
SELECT sum(orders.price)
FROM orders, customers, suppliers,product,clerks
WHERE orders.custid = customers.custid
   AND customers.zipcode = 94025
   AND orders.suppid = suppliers.suppid
   AND suppliers.name = 'Johnson'
   AND orders.prodid = product.prodid
   AND product.type = 'hard drive'
   AND orders.clerkid = clerks.clerkid
   AND clerks.dept = 'Direct Sales'
```

Tento dotaz využívá obvyklé spojení typu hvězda, ve kterém je tabulka faktů spojena se všemi rozměrovými tabulkami pomocí cizího klíče. Každá rozměrová tabulka má selektivní filtr tabulky.

Optimální plán spojení typu hvězda je provést kartézský součin čtyř rozměrových tabulek a následně výsledek spojit s tabulkou faktů. Následující index tabulky faktů umožňuje optimalizátoru zvolit optimální plán dotazů:

```
CREATE INDEX ON orders(custid,suppid,prodid,clerkid)
```

Bez tohoto indexu by optimalizátor mohl zvolit nejprve spojení tabulky faktů s jednou rozměrovou tabulkou a následné spojení výsledku se zbývajícími rozměrovými tabulkami. Optimální plán poskytuje lepší výkon.

Konfigurace informací o prohledávání B-stromu za účelem zlepšení zpracování transakcí

Je-li v protokolované databázi provedena operace odstranění nebo aktualizace, není odpovídající položka indexu ihned odstraněna. Namísto toho je odpovídající položka indexu označena jako odstraněná, dokud jednotkový proces prohledávání B-stromu neprohledá index a neodstraní odstraněné položky. Index obsahující mnoho položek může způsobit závažné problémy s výkonem, protože před nalezením první platné položky je nutné prohledat velký počet položek.

V závislosti na používané aplikaci a na pořadí, ve kterém systém do indexu přidává klíče a odstraňuje je z indexu, se může stát, že struktura indexu bude nepoužitelná.

Prohledávání B-stromu zlepšuje zpracován transakcí protokolovaných databází, pokud jsou z tabulky s indexy odstraněny řádky. Na základě seznamu priorit určuje prohledávání B-stromu automaticky, které oddíly indexu budou vyčištěny.

Počet jednotkových procesů prohledávání B-stromu lze konfigurovat na libovolné kladné číslo. Za běhu můžete aktivitu prohledávání B-stromu vypnout příkazem **onmode -C**, který zastaví všechny jednotkové procesy prohledávání B-stromu. Jeden jednotkový proces prohledávání B-stromu bude vždy čistit samotný oddíl indexu, budete-li tedy mít příležitostně nebo trvale vyšší počet oddílů indexu, které vyžadují čištění, mohli byste chtít použít více jednotkových procesů prohledávání B-stromu.

Můžete také konfigurovat prahovou hodnotu, tj. minimální počet odstraněných položek, kterého musí index dosáhnout, aby mohl být vložen do seznamu priorit pro způsobilost k prohledávání a čištění jednotkovým procesem prohledávání B-stromu. Zvýšíte-li například prahovou hodnotu nad 5000, mohli byste zabránit častým aktivitám prohledávání B-stromu s indexy, které jsou přijímají nejvíce aktualizací a odstranění.

Jednotkové procesy prohledávání B-stromu pracují v těchto třech režimech:

- *Režim prohledávání na úrovni listu*. V tomto režimu se vyhledávají odstraněné položky indexu úplným prohledáváním na úrovni listu.

- *Režim prohledávání podle rozsahu*. Oddíl indexu má záznam o nejnižší a nejvyšší pozici, ve které byly nalezeny odstraněné položky. V režimu prohledávání podle rozsahu je oddíl indexu prohledáván na úrovni listu pouze v tomto rozsahu, a může tedy ušetřit mnoho nepotřebných operací čtení. Server provádí odlehčená prohledávání, která hned nepoužívají a nepřetěžují společnou oblast vyrovnávací paměti, dokonce ani tehdy, je-li čištění prováděno přes společnou oblast vyrovnávací paměti. Tento režim prohledávání je pro indexy, které přijímají operace odstranění pouze v určité oblasti, vhodnější než režim prohledávání na úrovni listu.
- *Režim alice (adaptive linear index cleaning)*. Je-li povoleno prohledávání alice, obdrží každý oddíl indexu bitovou mapu, která zaznamená, kde byla odstraněná položka v indexu nalezena. Prohledávání potom vyloučí všechny části indexu, ve kterých nebyly nalezeny žádné odstraněné položky. Počáteční velikost a granularita těchto bitových map závisí na velikosti oddílů, které reprezentují, a na aktuální úrovni prohledávání alice v celém systému. Server pravidelně kontroluje efektivnost každé bitové mapy podle pdílu stran, které mají být vyčištěny, a v případě potřeby upravuje prohledávání tak, aby mohl získat vhodnější informace.

****** až sem ES, odsud MSTA***** K určení počátečního počtu jednotkových procesů prohledávání B-stromu použijte konfigurační parametr BTSCANNER. U systémů s velkým množstvím velkých indexů nastavte pomocí volby rangesize konfiguračního parametru BTSCANNER prahovou hodnotu minimální velikosti oddílu pro prohledávání podle rozsahu.

Pokud chcete povolit režim prohledávání podle rozsahu, nastavte volbu rangesize konfiguračního paremetru BTSCANNER na minimální velikost oddílu, který má být v tomto režimu prohledán. Zadejte velikost v kilobajtech. Chcete-li umožnit prohledávání malých indexů pomocí metody prohledávání na úrovni listů, nastavte volbu rangesize na hodnotu 100.

Příklad nastavení režimu prohledávání podle rozsahu:

```
BTSCANNER num=2,threshold=50000,rangesize=100
```

Chcete-li povolit režim alice, nastavte volbu alice na hodnotu v rozsahu 1 až 12 (nejjemnější počáteční granularita). U malých až středně velkých systémů s několika málo nebo žádnými indexy většími než 1 GB nastavte volbu alice na hodnotu 6 nebo 7. U systémů s velkými indexy nastavte volbu alice na vyšší režim.

Příklad nastavení režimu prohledávání podle rozsahu:

```
BTSCANNER num=2,threshold=5000,rangesize=100,alice=5
```

Další informace o konfiguračním parametru BTSCANNER a o tom, jak databázový server spravuje strom indexů, naleznete v kapitole týkající se konfiguračních parametrů a v kapitole týkající se struktury disku a úložišť v příručce *IBM Informix Dynamic Server Administrator's Reference*.

K monitorování aktivit prohledávání B-stromu použijte volbu **onstat -C**.

Ke změně konfigurace prohledávání B-stromu při provozu použijte volbu **onmode -C**.

Další informace o volbách **onstat -C** a **onmode -C** naleznete v příručce *IBM Informix Dynamic Server Administrator's Reference* ****** KONEC MSTA.

Určení množství volného místa ve stránce indexu

K určení množství volného místa v každé stránce indexu použijte příkaz **oncheck -pT** . Pokud má tabulka poměrně nízkou aktivitu aktualizací a existuje velké množství volného

místa, pravděpodobně budete chtít index vypustit a opětovně vytvořit s větší hodnotou volby FILLFACTOR a dát tak k dispozici nevyužité volné místo na disku.

Zvýšení výkonu distribuovaných dotazů

Optimalizátor předpokládá, že přístup k řádku ve vzdálené databázi bude trvat déle než přístup k řádku v místní databázi. Odhady optimalizátoru zahrnují nákladovost na načtení řádku z disku a jeho přenos v síti. Příklad vyšší odhadnuté nákladovosti naleznete v části "Zobrazení plánu dotazů pro distribuovaný dotaz" na stránce 13-18.

Ukládání přenosů dat distribuovaných dotazů do vyrovnávací paměti

Databázový server určuje velikost vyrovnávací paměti k odesílání a přijímání dat do a ze vzdáleného serveru pomocí následujících faktorů:

- Velikost řádků

  Databázový server vypočítá velikost řádku sečtením průměrné velikosti přesunutí (pokud je dostupná) nebo délky (z tabulky systémového katalogu **syscolumns**) sloupců.

- nastavení **FET_BUF_SIZE** na klienta

  Velikost a počet přenosů dat je možné snížit použitím proměnné prostředí **FET_BUF_SIZE** ke zvýšení velikosti vyrovnávací paměti, kterou databázový server používá k odesílání a přijímání řádků do a ze vzdáleného databázového serveru.

  Minimální velikost vyrovnávací paměti je 1024 nebo 2048 bajtů v závislosti na velikosti řádků. Pokud je velikost řádků větší než 1024 nebo 2048, databázový server použije hodnotu proměnné prostředí **FET_BUF_SIZE**.

  Další informace o proměnné prostředí **FET_BUF_SIZE** naleznete v příručce *IBM Informix Guide to SQL: Reference*.

Zobrazení plánu dotazů pro distribuovaný dotaz

Obrázek 13-5 zobrazuje plán dotazů zvolený pro distribuovaný dotaz.

```
QUERY:
------
select l.customer_num, l.lname, l.company,
         l.phone, r.call_dtime, r.call_descr
       from customer l, vjp_stores@gilroy:cust_calls r
        where l.customer_num = r.customer_num

 Estimated Cost: 9
Estimated # of Rows Returned: 7

  1) informix.r:  REMOTE PATH

  2) informix.l: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.l.customer_num = informix.r.customer_num
NESTED LOOP JOIN
```

Obrázek 13-5. Vybraný výstup příkazu SET EXPLAIN ALL pro distribuovaný dotaz, část 3

Následující tabulka zobrazuje hlavní rozdíly mezi zvolenými plány dotazů distribuovaného spojení a místního spojení.

Výstupní řádek v obrázku Obrázek 13-5 pro distribuovaný dotaz	Výstupní řádek v obrázku Obrázek 11-3 pro pouze místní dotaz	Popis rozdílu
**********LŠ vjp_stores@gilroy: virginia.cust_calls	informix.cust_calls	Název vzdálené tabulky je uváděn s názvem databáze a serveru.
Estimated Cost: 9	Estimated Cost: 7	Optimalizátor odhaduje u distribuovaného dotazu vyšší nákladovost.
informix.r: REMOTE PATH	informix.r: SEQUENTIAL SCAN	Optimalizátor volí výběr vnější vzdálené tabulky **cust_calls** na vzdáleném serveru.
select x0.call_dtime,x0.call_descr,x0. customer_num from vjp_stores:"virginia".cust_ calls x0		Příkaz jazyka SQL, který databázový server odešle vzdálenému serveru. Vzdálený server opětovnou optimalizací tohoto příkazu zvolí skutečný plán.

Zlepšení sekvenčního prohledávání

Pokud chcete zvýšit výkon operací sekvenčního čtení ve velkých tabulkách, vyhněte se opakovaný sekvenčním prohledáváním.

Sekvenční přístup k jiné tabulce, než je první tabulka v plánu, je nebezpečný, protože hrozí čtením každého řádku tabulky pro každý řádek vybraný z předchozích tabulek. Měli byste být schopni rozhodnout, o jaké číslo se jedná: možná několik, ale možná také stovky nebo dokonce tisíce.

Pokud je tabulka malá, není její opakované čtení nebezpečné, protože celá tabulka zůstává v paměti. Sekvenční prohledávání tabulky v paměti může být rychlejší než prohledávání stejné tabulky pomocí indexu, zvláště pokud správa těchto stránek indexů v paměti vytlačí z vyrovnávací paměti jiné užitečné stránky.

Pokud je tabulka větší než několik stránek, opakovaný sekvenční přístup zapříčiní nízký výkon. Jedním způsobem, jak zabránit problémům, je poskytnutí indexu sloupci, který je použit ke spojení tabulky.

Každý uživatel s oprávněním zdroje může vytvářet další indexy. K vytvoření indexu použijte příkaz CREATE INDEX.

Index spotřebovává místo na disku, které odpovídá šířce klíčových hodnot a počtu řádků. (Další informace naleznete v části "Odhad indexových stránek" na stránce 7-1.) Databázový server také musí index aktualizovat při každém vložení, odstranění nebo aktualizaci řádků; aktualizace indexu tyto operace zpomaluje. Pokud je to nutné, můžete po řadě dotazů index uvolnit pomocí příkazu DROP INDEX, což uvolní místo a zjednoduší aktualizace tabulky.

Povolení skládání pohledů za účelem zvýšení výkonu dotazů

Výkon dotazu, který obsahuje pohled, můžete zvýšit povolením skládání pohledů pomocí konfiguračního parametru IFX_FOLDVIEW. Pokud je konfigurační parametr IFX_FOLDVIEW povolen, pohledy jsou skládány do nadřazeného dotazu. Protože pohledy jsou skládány do nadřazeného dotazu, výsledky dotazu nejsou umisťovány do dočasné tabulky.

Skládání pohledů je možné použít pro následující typy dotazů:

- Pohledy, které obsahují klauzuli UNION ALL a nadřazený dotaz má pravidelné spojení, spojení Informix, spojení ANSI nebo klauzuli ORDER BY.
- Pohledy se spojeními vícenásobných tabulek, kde hlavní dotaz obsahuje vnější spojení typu Informix nebo ANSI.

Ke skládání pohledů nedojde u následujících typů dotazů provádějících operaci UNION ALL, která zahrnuje pohled:

- Pohled obsahuje jednu z následujících klauzulí: AGGREGATE, GROUP BY, ORDER BY, UNION, DISTINCT nebo OUTER JOIN (buď typu Informix nebo ANSI).
- Nadřazený dotaz obsahuje klauzuli UNION nebo UNION ALL.

V těchto situacích je vytvořena dočasná tabulka, která obsahuje výsledky dotazu.

Postup povolení skládání pohledů:

1. Nastavte konfigurační parametr IFX_FOLDVIEW na hodnotu 1.

Snížení vlivu operací spojení a řazení

Jakmile porozumíte tomu, co dotaz dělá, hledejte způsoby, jak získat s menším úsilím stejný výstup. Následující návrhy mohou pomoci s efektivnějším přepsáním dotazů:

- Vyhněte se operacím řazení nebo je zjednodušte.
- Používejte paralelní řazení.
- Snižte rozsah řazení použitím dočasných tabulek.

Vyhnutí se nebo zjednodušení operací řazení

Řazení není vždy nutné. Algoritmus řazení je vysoce vyladěný a extrémně efektivní. Je stejně rychlý jako kterýkoliv program řazení, který je možné použít na stejná data. Nemusíte se vyhýbat občasným tříděním nebo tříděním malého počtu výstupních řádků.

Pokuste se vyhnout nebo snížit rozsah opakovaných řazení velkých tabulek. Optimalizátor se vyhýbá řazení, kdykoliv může k automatickému vytvoření výstupu ve správném pořadí použít index. V použití indexu brání optimalizátoru následující faktory:

- Jeden nebo více seřazených sloupců není v indexu zahrnuto.
- Sloupce jsou v indexu pojmenovány v odlišné posloupnosti než v klauzuli ORDER BY nebo GROUP BY.
- Seřazené sloupce jsou pořizovány z různých tabulek.

Další způsob, jak se vyhnout řazení, naleznete v části “Použití dočasných tabulek ke snížení rozsahu řazení” na stránce 13-21.

Pokud je řazení nutné, hledejte cesty, jak jej zjednodušit. Jak je diskutováno v části “Časová nákladovost řazení” na stránce 10-22. Řazení je rychlejší, pokud můžete řadit méně sloupců nebo užší sloupce.

Použití paralelního řazení

Pokud se nemůžete vyhnout řazení, databázový server využije zdroje více procesorů k paralelnímu provedení operací řazení a sloučení. Databázový server může použít paralelní řazení pro všechny dotazy; paralelní řazení nejsou omezena dotazy PDQ. Pokud chcete ovládat počet jednotkových procesů, které databázový server použije k řazení řádků, použijte proměnnou prostředí **PSORT_NPROCS**.

Pokud je priorita PDQ větší než 0 a proměnná prostředí **PSORT_NPROCS** je větší než 1, dotaz využije paralelní řazení i vlastnosti PDQ, jako je paralelní prohledávání a přídavná

paměť. Uživatelé mohou pomocí proměnné prostředí **PDQPRIORITY** požádat o specifickou část zdrojů PDQ pro dotaz. Pomocí parametru MAX_PDQPRIORITY můžete počet takovýchto požadavků uživatelů omezit. Další informace naleznete v části “Omezení zdrojů PDQ v dotazech pomocí konfiguračního parametru MAX_PDQPRIORITY” na stránce 3-10.

V některých případech může množství řazených dat přetéci zdroje paměti přidělené dotazu, což povede k uložení vstupu - výstupu do prostoru dbspace nebo souboru řazení. Další informace naleznete v části “Konfigurace prostorů dbspace pro dočasné tabulky a soubory řazení” na stránce 5-7.

Použití dočasných tabulek ke snížení rozsahu řazení

Vytvoření dočasné, seřazené části tabulky může urychlit dotaz. To může pomoci předejít operacím vícenásobného řazení a může jinými způsoby zjednodušit činnost optimalizátoru.

Předpokládejte například, že aplikace vytvoří řadu zpráv o zákaznících, kteří mají výjimečné účetní bilance, jednu zprávu pro každou hlavní poštovní oblast, seřazenou podle jména zákazníka. Jinými slovy se vyskytne řada dotazů, každý v následující formě (s využitím fiktivní tabulky a názvů sloupců):

```
SELECT cust.name, rcvbles.balance, ...other columns...
   FROM cust, rcvbles
   WHERE cust.customer_id = rcvbles.customer_id
      AND rcvbls.balance > 0
      AND cust.postcode LIKE '98_ _ _'
   ORDER BY cust.name
```

Tento dotaz přečte celou tabulku **zak**. Pro každý řádek určeného poštovního směrovacího čísla databázový server prohledá index sloupce **rcvbles.customer_id** a pro každou shodu provede nesekvenční přístup na disk. Řádky jsou zapsány do dočasného souboru a seřazeny. Další informace o dočasných souborech naleznete v části “Konfigurace prostorů dbspace pro dočasné tabulky a soubory řazení” na stránce 5-7.

Tato procedura je přijatelná, pokud je dotaz prováděn pouze jednou, ale tento příklad zahrnuje řady dotazů, z nichž každý způsobí stejné množství práce.

Alternativou je vybrání všech zákazníků s výjimečnými bilancemi do dočasné tabulky, seřazené podle jména zákazníka, jak uvádí následující příklad:

```
SELECT cust.name, rcvbles.balance, ...ostatní sloupce...
   FROM cust, rcvbles
   WHERE cust.customer_id = rcvbles.customer_id
      AND cvbls.balance > 0
   INTO TEMP cust_with_balance
```

Poté můžete provádět dotazy na dočasnou tabulky, jak uvádí následující příklad:

```
SELECT *
   FROM cust_with_balance
   WHERE postcode LIKE '98_ _ _'
   ORDER BY cust.name
```

Každý dotaz čte dočasnou tabulku sekvenčně, ale tabulka má méně řádků.

Více paměti pro dotazy se hashovacími spojeními, souhrny a dalšími prvky náročnými na paměť

Pokud chcete zvětšit množství paměti dostupné pro dotazy jiné než PDQ a priorita PDQ je nastavena na hodnotu 0 (nula), můžete ke změně množství paměti použít kteroukoli z následujících voleb:

- konfigurační parametr DS_NONPDQ_QUERY_MEM,

- příkaz **onmode -wm** nebo **onmode -wf** ,
- Volba **Non PDQ Query Memory** v nabídce programu ON-Monitor **pdQ**

Pokud používáte například obslužný program onmode, určete hodnotu tak, jak uvádí následující příklad:

```
onmode -wf DS_NONPDQ_QUERY_MEM=500
```

Minimální hodnota parametru DS_NONPDQ_QUERY_MEM je 128 KB. Maximální podporovaná hodnota je 25 procent hodnoty konfiguračního parametru DS_TOTAL_MEMORY. Výchozí hodnota parametru DS_NONPDQ_QUERY_MEM je 128 KB. Pokud určujete hodnotu parametru DS_NONPDQ_QUERY_MEM, určete a upravte hodnotu podle počtu a velikosti řádků tabulky, které jsou obsaženy v dotazu.

Hodnota parametru DS_NONPDQ_QUERY_MEM je vypočítána během spuštění databázového serveru podle vypočítané hodnoty parametru DS_TOTAL_MEMORY.

Pokud během zpracování konfiguračního parametru DS_NONPDQ_QUERY_MEM databázový server změní hodnotu, kterou jste nastavili, odešle server zprávu v tomto formátu:

DS_NONPDQ_QUERY_MEM recalculated and changed from stará_hodnota Kb to nová_hodnota Kb.

Hodnota stará_hodnota představuje hodnotu, kterou jste přiřadili ke konfiguračnímu parametru DS_NONPDQ_QUERY_MEM v uživatelském konfiguračním souboru a hodnota nová_hodnota představuje hodnotu, kterou určil databázový server.

Optimalizace doby odezvy uživatele u dotazů

Následující části popisují, jak je možné ovlivnit čas optimalizace dotazu a čas, během kterého jsou řádky vráceny uživateli.

Úroveň optimalizace

Optimální celkový výkon běžně získáte s výchozí úrovní optimalizace, HIGH (vysoká). Čas, který trvá optimalizace příkazu, obvykle není důležitý. Pokud však experimentování s aplikací odhalí, že dotaz trvá stále příliš dlouho, můžete úroveň optimalizace nastavit na hodnotu LOW (nízká) a poté zkontrolovat výstup příkazu SET EXPLAIN, zda optimalizátor zvolil stejný plán dotazů jako předtím.

Pokud chcete určit úroveň optimalizace databázového serveru na hodnotu HIGH nebo LOW, použijte příkaz SET OPTIMIZATION. Podrobný popis tohoto příkazu naleznete v příručce *IBM Informix Guide to SQL: Syntax*.

Cíl optimalizace

Existují dva následující typy cílů optimalizace výkonu dotazů:

- optimalizace celkového času dotazu
- optimalizace doby odezvy uživatele

Celkový čas dotazu je čas, který trvá vrácení všech řádků aplikaci. Celkový čas dotazu je nejdůležitější pro dávkové zpracování nebo pro dotazy, které vyžadují, aby před vrácením výsledku uživateli byly zpracovány všechny řádky, jak uvádí následující dotaz:

```
SELECT count(*) FROM orders
WHERE order_amount > 2000;
```

Doba odezvy uživatele je čas, který databázovému serveru trvá vrácení obrazovky s řádky interaktivní aplikaci. V interaktivních aplikacích může být najednou vrácena pouze úplná obrazovka dat. Uživatelská aplikace může například v případě následujícího dotazu zobrazit najednou pouze 10 řádků:

```
SELECT * FROM orders
WHERE order_amount > 2000;
```

Cíl optimalizace, který je důležitější, může mít vliv na cestu dotazu, kterou optimalizátor zvolí. Optimalizátor může při provádění dotazu zvolit například spojení s vnořenou smyčkou namísto hashovacího spojení, pokud je doba odezvy uživatele nejdůležitější, ačkoliv hashovací spojení může vést ke snížení celkového času dotazu.

Určení cíle výkonu dotazu

Ve výchozím režimu optimalizátor zvolí plány dotazů, které optimalizují celkový čas dotazu. Optimalizaci doby odezvy uživatele můžete určit pomocí několika různých úrovní:

- Pro systém databázového serveru

  Pokud chcete optimalizovat dobu odezvy uživatele, nastavte konfigurační parametr OPT_GOAL na hodnotu 0, jak uvádí následující ukázky příkladů:

  ```
  OPT_GOAL 0
  ```

  Nastavením konfiguračního parametru OPT_GOAL na hodnotu -1 optimalizujete celkový čas dotazu.

- Pro prostředí uživatele

  Proměnná prostředí **OPT_GOAL** může být nastavena před spuštěním uživatelské aplikace.

  Jen pro UNIX

  Pokud chcete optimalizovat dobu odezvy uživatele, nastavte proměnnou prostředí **OPT_GOAL** na hodnotu 0, jak uvádí následující ukázky příkazů:

  ```
  Prostředí Bourne                OPT_GOAL = 0
                export OPT_GOAL

  Prostředí C               setenv OPT_GOAL 0
  ```

  Konec Jen pro UNIX

  Pokud chcete optimalizovat celkový čas dotazu, nastavte proměnnou prostředí **OPT_GOAL** na hodnotu -1.

- v rámci relace

  Cíl optimalizace je možné v jazyce SQL ovládat příkazem SET OPTIMIZATION. Cíl optimalizace nastavený tímto příkazem je platný, dokud relace neskončí nebo dokud není cíl optimalizace změněn dalším příkazem SET OPTIMIZATION.

  Následující příkaz způsobí, že optimalizátor zvolí plány dotazů, které příznivě ovlivní optimalizaci celkového času dotazu:

  ```
  SET OPTIMIZATION ALL_ROWS
  ```

  Následující příkaz způsobí, že optimalizátor zvolí plány dotazů, které příznivě ovlivní optimalizaci doby odezvy uživatele:

  ```
  SET OPTIMIZATION FIRST_ROWS
  ```

- Pro jednotlivé dotazy

  Pokud chcete instruovat optimalizátor, který cíl dotazu má použít, použijte direktivy optimalizátoru FIRST_ROWS a ALL_ROWS. Další informace o těchto direktivách naleznete v části “Direktivy cílů optimalizace” na stránce 11-7.

Pořadí těchto úrovní je následující:

- direktivy optimalizátoru
- příkaz SET OPTIMIZATION
- proměnná prostředí **OPT_GOAL**
- konfigurační parametr OPT_GOAL

Direktivy optimalizátoru mohou mít přednost například před cílem, který určuje příkaz SET OPTIMIZATION.

Upřednostňované plány dotazů pro optimalizaci doby odezvy uživatele

Pokud optimalizátor zvolí plán dotazů, který povede k optimalizaci doby odezvy uživatele, vypočítá nákladovost na získání prvního řádku dotazu každého plánu a zvolí plán s nejnižší nákladovostí. Plán, který získal první řádek s nejnižší nákladovostí, nemusí být v některých případech optimální cestou k získání všech řádků dotazu.

Následující části vysvětlují některé možné rozdíly v plánech dotazů.

Spojení s vnořenou smyčkou a hashovací spojení: hashovací spojení mají obvykle vyšší nákladovost na získání prvního řádku než spojení s vnořenou smyčkou. Databázový server musí před získáním řádků vytvořit tabulku hashovací funkce. V některých případech je však při použití hashovacího spojení kratší celkový čas dotazu.

Tabulka **tab2** v následujícím příkladu má index na sloupci **sl1**, zatímco tabulka **tab1** index na sloupci **sl1** nemá. Při provedení příkazu SET OPTIMIZATION ALL_ROWS před spuštěním dotazu použije databázový server hashovací spojení a ignoruje existující index, jak uvádí následující výstup vybraného příkazu SET EXPLAIN:

```
QUERY:
------
SELECT * FROM tab1,tab2
WHERE tab1.sl1 = tab2.sl1
Estimated Cost: 125
Estimated # of Rows Returned: 510
1) lsuto.tab2: SEQUENTIAL SCAN
2) lsuto.tab1: SEQUENTIAL SCAN
DYNAMIC HASH JOIN
    Dynamic Hash Filters: lsuto.tab2.sl1 = lsuto.tab1.sl1
```

Pokud však provedete příkaz SET OPTIMIZATION FIRST_ROWS před spuštěním dotazu, databázový server použije spojení s vnořenou smyčkou. Klauzule (FIRST_ROWS OPTIMIZATION) v následující části výstupu příkazu SET EXPLAIN ukazuje, že optimalizátor použil pro dotaz optimalizaci doby odezvy uživatele:

```
QUERY:        (FIRST_ROWS OPTIMIZATION)
------
SELECT * FROM tab1,tab2
WHERE tab1.sl1 = tab2.sl1
Estimated Cost: 145
Estimated # of Rows Returned: 510
1) lsuto.tab1: SEQUENTIAL SCAN
2) lsuto.tab2: INDEX PATH
    (1) Index Keys: sl1
        Lower Index Filter: lsuto.tab2.sl1 = lsuto.tab1.sl1
NESTED LOOP JOIN
```

Prohledávání tabulek a prohledávání indexů: V případech, kdy databázový server vrací velké množství řádků tabulky, volba nižší nákladovosti pro cíl celkového času dotazu může znamenat prohledávání tabulky namísto použití indexů. Pokud však chcete získat první řádek, volba nižší nákladovosti pro cíl nižší odezvy uživatele může při přístupu k tabulce vést k použití indexů.

Uspořádání pomocí fragmentovaných indexů: Pokud index není fragmentován, databázový server jej může využít a vyhnout se tak řazení. Další informace o možnostech, jak se vyhnout řazením, naleznete v části “Vyhnutí se nebo zjednodušení operací řazení” na stránce 13-20. Pokud je však index fragmentován, seřazení může být zaručeno pouze v rámci fragmentu, ne mezi fragmenty.

Nejméně nákladnou volbou pro cíl celkového času dotazu je obvykle paralelní prohledání fragmentů a následné použití paralelního řazení, které vytvoří odpovídající pořadí. Tato volba však není vhodná pro cíl doby odezvy uživatele.

Pokud je namísto toho důležitá doba odezvy uživatele, databázový server přečte paralelně všechny fragmenty indexů a sloučí data ze všech fragmentů. Obecně není nutné další řazení.

Optimalizace dotazů pro uživatelské datové typy

Dotazy, které přistupují k uživatelským datovým typům (UDT) mohou využívat funkce stejného výkonu, které vestavěné datové typy používají:

- indexy

  Pokud dotaz přistupuje k malému počtu řádků, index urychlí vyhledávání, protože databázový server nemusí při hledání řádků číst všechny stránky. Další informace naleznete v části “Indexy v uživatelských datových typech” na stránce 7-15.
- paralelní databázové dotazy (PDQ)

  Dotazy, které přistupují k uživatelským datům mohou využít paralelní prohledávání a paralelní provádění. Další informace o paralelním provádění uživatelských rutin naleznete v kapitole věnované výkonu v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

  Pokud chcete spustit paralelní provádění dotazu, nastavte proměnnou prostředí **PDQPRIORITY** nebo použijte příkaz jazyka SQL SET PDQPRIORITY. Další informace o nastavení priority PDQ a konfiguračních parametrů, které prioritu PDQ ovlivňují, naleznete v části “Přidělování zdrojů paralelním databázovým dotazům” na stránce 12-7.
- direktivy optimalizátoru

Kromě toho mohou programátoři napsat následující funkce nebo uživatelské rutiny, které optimalizátoru pomohou vytvořit pro dotazy efektivní plán dotazů:

- paralelní uživatelské rutiny, které mohou využít paralelní databázové dotazy
- uživatelské funkce selektivity, které vypočítají očekávaný zlomek řádků, který je pro funkci vhodný
- uživatelské funkce nákladovosti, které vypočítávají očekávanou relativní nákladovost provádění uživatelské rutiny
- uživatelské statistické funkce, které může příkaz UPDATE STATISTICS použít ke generování statistických údajů a distribucí dat
- uživatelské funkce negace, které umožňují optimalizátoru více voleb

Tyto techniky shrnují následující části. Úplnější popis postupu při zápisu a registraci uživatelských funkcí selektivity a uživatelských funkcí nákladovosti naleznete v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Paralelní uživatelské rutiny

Jedním způsobem, jak provádět uživatelské rutiny je vyjádření v dotazu. Paralelní provádění můžete využít, pokud je uživatelská rutina ve výrazu nebo jedné z následujících částí dotazu:

- Klauzule WHERE
- Seznam SELECT

- Seznam GROUP BY
- Přetížený operátor porovnání
- Uživatelský agregát
- Klauzule HAVING
- Seznam výběru pro příkaz paralelního vkládání
- Prohledávání indexů obecných B-stromů ve fragmentech indexů více za předpokladu, že funkce porovnávání použitá v prohledávání indexů B-stromů je paralelizovatelná

Předpokládejme například, že vytvoříte netransparentní datový typ **circle**, tabulku **circ_t**, která určuje sloupec typu **circle**, uživatelskou rutinu **area** a poté provedete následující ukázkový dotaz:

```
SELECT circle, area(circle)
   FROM cir_t
   WHERE circle > "(6,2,4)";
```

V tomto ukázkovém dotazu jsou prováděny paralelně následující operace:

- Uživatelská rutina **area(circle)** v seznamu SELECT

  Pokud je tabulka **circ_t** fragmentovaná, vícenásobné uživatelské rutiny **area** mohou být prováděny paralelně, jedna uživatelská rutina na každý fragment.
- Výraz circle > ″(6,2,4)″ v klauzuli WHERE

  Pokud je tabulka **circ_t** fragmentovaná, vícenásobné prohledávání tabulky může být prováděno paralelně, jedno prohledávání na každý fragment. Vícenásobné operátory porovnávání “>” mohou být poté prováděny paralelně, jeden operátor na fragment.

Uživatelská rutina není při výchozím nastavení prováděna paralelně. Pokud chcete povolit paralelní provádění uživatelských rutin, je nutné provést následující kroky:

- Určete modifikátor PARALLELIZABLE v příkazu CREATE FUNCTION nebo ALTER FUNCTION.
- Ujistěte se, že uživatelská rutina nevolá funkce, které nejsou priority PDQ bezpečné z hlediska jednotkových procesů.
- Zapněte prioritu PDQ.
- Použijte v paralelním databázovém dotazu uživatelskou rutinu.

Funkce selektivity a nákladovosti

Příkaz CREATE FUNCTION umožňuje uživatelům vytvářet uživatelské rutiny. Uživatelskou rutinu je možné umístit do příkazu jazyka SQL, jak uvádí následující příklad:

```
SELECT * FROM obrázek
WHERE get_x1(obrazek.ob2) and get_x2(obrazek.ob1)
```

Optimalizátor nemůže přesně vyhodnotit nákladovost provedení uživatelské rutiny bez dalších informací. Optimalizátoru můžete poskytnout nákladovost a selektivitu funkce. Databázový server použije nákladovost a selektivitu společně k určení nejlepší cesty. Další informace o selektivitě naleznete v části “Filtry s uživatelskými rutinami” na stránce 13-3.

V předchozím příkladu nemohl optimalizátor určit, kterou funkci má provést jako první, zda funkci **get_x1** nebo **get_x2**. Pokud je provedení funkce nákladné, administrátor databáze může funkci přiřadit vyšší nákladovost nebo selektivitu, což může optimalizátor přimět, aby změnil plán dotazů a dosáhnul tak vyššího výkonu. Pokud by provedení funkce **get_x1** v předchozím příkladu bylo nákladnější, administrátor databáze (DBA) může této funkci přiřadit vyšší nákladovost, což může způsobit, že optimalizátor provede jako první funkci **get_x2**.

K příkazu CREATE FUNCTION je možné přidat následující modifikátory rutin a změnit tak nákladovost nebo selektivitu, kterou optimalizátor přiřazuje k funkci:

- **selfunc=***název_funkce*
- **selconst=***celé_číslo*
- **costfunc=***název_funkce*
- **percall_cost=***celé_číslo*

Další informace o modifikátorech nákladovosti a selektivity naleznete v příručce *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Uživatelské statistické údaje uživatelských datových typů (UDT)

Protože databázový server nemá k dispozici informace o vlastnostech a použití uživatelských datových typů (UDT), nemůže u uživatelských datových typů shromažďovat informace o distribucích nebo sloupcích **colmin** a **colmax** (nacházejí se v tabulce systémového katalogu **syscolumns**). Namísto toho můžete vytvořit speciální funkci, která naplní tyto statistické údaje daty. Databázový server spouští po provedení příkazu UPDATE STATISTICS funkci shromažďování statistických údajů.

Další informace o významu aktualizace statistických údajů naleznete v části “Statistické údaje uchovávané pro tabulku a index” na stránce 10-18. Informace o zvýšení výkonu naleznete v části “Aktualizace statistických údajů sloupců s uživatelskými datovými typy” na stránce 13-12.

Funkce negace

Funkce negace přebírá stejné argumenty jako její protější funkce a ve stejném pořadí, ale vrací booleovský doplněk. To znamená, že pokud funkce vrací pro dané argumenty hodnotu TRUE, její funkce negace vrací hodnotu FALSE, pokud jsou jí předány stejné argumenty a ve stejném pořadí. V některých případech může databázový server zpracovat dotaz mnohem efektivněji, pokud je smysl dotazu převrácen. To znamená, že otázka ″Je **x** větší než **y**?″ se změní na otázku ″Je **y** menší nebo rovno **x**?″

Mezipaměť příkazů SQL

Než databázový server provede příkaz jazyka SQL, musí příkaz nejprve analyzovat a optimalizovat. Tyto kroky mohou zabírat čas v závislosti na velikosti příkazu jazyka SQL.

Databázový server může analyzovaný a optimalizovaný příkaz jazyka SQL uložit do virtuální části sdílené paměti v oblasti, která se nazývá *mezipaměť pro příkazy* jazyka SQL. K mezipaměti příkazů SQL mohou přistupovat všichni uživatelé a všichni uživatelé mohou obejít kroky analýzy a optimalizace před provedením dotazu. Tato vlastnost může vést k následujícím významným zvýšením výkonu:

- Doby odezvy při provádění příkazů jazyka SQL uživateli jsou kratší.

  Příkazy jazyka SQL, jejichž optimalizace trvá dlouho (obvykle protože obsahují mnoho tabulek a filtrů v klauzuli WHERE), se z mezipaměti příkazů SQL provádějí rychleji, protože databázový server nemusí příkazy analyzovat nebo optimalizovat.
- Dochází ke snížení využití paměti, protože databázový server sdílí datové struktury dotazů mezi uživateli.

  Snížení využití paměti díky mezipaměti příkazů SQL je vyšší, pokud má příkaz v seznamu výběru mnoho názvů sloupců.

Další informace o efektu mezipaměti příkazů SQL na celkový výkon systému naleznete v části “Mezipaměť příkazů SQL” na stránce 4-24.

Kdy použít mezipaměť příkazů SQL

Aplikace mohou výhodně využít mezipaměť příkazů SQL, pokud více uživatelů provádí stejné příkazy jazyka SQL. Databázový server rozhodne, že příkazy jsou stejné, pokud všechny znaky přesně odpovídají. Pokud například 50 obchodních zástupců provádí v aplikaci v průběhu dne příkaz **add_order**, všichni provádí stejné příkazy jazyka SQL, pokud aplikace obsahuje příkazy jazyka SQL, které používají hostitelské proměnné podobně, jako uvádí následující příklad:

```
SELECT * FROM ORDERS WHERE order_num = :hostvar
```

Tento druh aplikace využívá mezipaměť příkazů SQL, protože uživatelé budou příkazy jazyka SQL pravděpodobněji zjišťovat v mezipaměti příkazů SQL.

Databázový server nebude brát v úvahu přesné shody následujících příkazů jazyka SQL, protože obsahují v klauzuli WHERE různé hodnoty literálů:

```
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
   AND order_date > "01/01/07"
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
   AND order_date > "01/01/2007"
```

Výkon se s mezipamětí příkazů SQL nezlepší v následujících situacích:

- Pokud se aplikace poskytující hlášení spouští jednou během noci a provádí příkazy jazyka SQL, které nepoužívají žádné další aplikace, pak tato aplikace mezipaměť pro příkazy nevyužije.
- Pokud aplikace připraví příkaz a poté jej mnohokrát provede, výkon se díky mezipaměti příkazů SQL nezvýší, protože příkaz je optimalizován právě během provádění příkazu PREPARE.

Pokud příkaz obsahuje hostitelské proměnné, databázový server nahradí při ukládání příkazu do mezipaměti příkazů SQL hostitelské proměnné zástupnými symboly. Proto je příkaz optimalizován, aniž by měl databázový server přístup k hodnotám hostitelských proměnných. V některých případech, pokud měl databázový server přístup k hodnotám hostitelských proměnných, je možné, že příkaz bude optimalizován odlišně, protože distribuce u sloupců obvykle informují optimalizátor o tom, kolik přesně řádků projde filtrem.

Pokud je provádění příkazu jazyka SQL, který obsahuje hostitelské proměnné, se zapnutou mezipamětípříkazů SQL pomalé, zkuste mezipaměť příkazů SQL vyprázdnit pomocí příkazu **onmode -e flush** a spustit dotaz s hodnotami, které jsou častěji používány mezi více provedeními dotazu. Když mezipaměť vyprázdníte, databázový server dotaz opětovně optimalizuje a vygeneruje plán dotazů, který je optimalizován pro tyto často používané hodnoty.

Důležité: Databázový server vyprázdní položku mezipaměti pro příkazy SQL, pouze pokud není používána. Pokud některá aplikace připraví příkaz a zachová jej, položka bude stále používána. V tomto případě je nutné, aby aplikace příkaz před použitím vyprázdnění uzavřela.

Použití mezipaměti příkazů SQL

Rozhodnutí, zda povolit mezipaměť příkazů SQL, je obvykle na administrátorovi databázového serveru. Pokud je mezipaměť příkazů SQL povolena, jednotliví uživatelé se mohou rozhodnout, zda mezipaměť příkazů SQL pro své specifické prostředí nebo aplikaci použijí nebo ne.

Při správě mezipaměti příkazů SQL dochází k zahlcení zpracování, uživatelé by proto neměli mezipaměť příkazů SQL používat, pokud žádní další uživatelé nesdílí v aplikaci příkazy jazyka SQL.

Následující části popisují, jak může administrátor databázového serveru povolit mezipaměť příkazů SQL v jednom ze dvou režimů:

- Vždy používat mezipaměť příkazů SQL, pokud uživatel výslovně neurčí, že nemá být použita.
- Používat mezipaměť příkazů SQL, pouze pokud uživatel výslovně určí, že má být použita.

Povolení mezipaměti příkazů SQL

Databázový server nebude používat mezipaměťpříkazů SQL, pokud bude konfigurační parametr STMT_CACHE nastaven na hodnotu 0 (výchozí hodnota).

Ke změně konfiguračního parametru STMT_CACHE na tuto výchozí hodnotu použijte jednu z následujících metod:

- Pokud chcete určit konfigurační parametr STMT_CACHE, aktualizujte soubor ONCONFIG a restartujte databázový server.

  Pokud nastavíte konfigurační parametr STMT_CACHE na hodnotu 1, databázový server použije mezipaměť příkazů SQL u jednotlivého uživatele, pokud uživatel nastaví proměnnou prostředí **STMT_CACHE** na hodnotu 1 nebo pokud v rámci aplikace provede příkaz SET STATEMENT CACHE ON.

  ```
  STMT_CACHE 1
  ```

  Pokud je konfigurační parametr STMT_CACHE nastaven na hodnotu 2, databázový server bude u všech uživatelů ukládat příkazy jazyka SQL do mezipaměti příkazů SQL, pokud jednotliví uživatelé nevypnou tuto funkci pomocí proměnné prostředí **STMT_CACHE** nebo pomocí příkazu SET STATEMENT CACHE OFF.

  ```
  STMT_CACHE 2
  ```

- Pokud chcete dynamicky potlačit konfigurační parametr STMT_CACHE, použijte příkaz **onmode -e**.

  Pokud použijete klíčové slovo **enable**, databázový server použije mezipaměť příkazů SQL pro jednotlivého uživatele, pokud uživatel nastaví proměnnou prostředí **STMT_CACHE** na hodnotu 1 nebo v rámci aplikace provede příkaz SET STATEMENT CACHE ON.

  ```
  onmode -e enable
  ```

  Pokud použijete klíčové slovo **on** databázový server bude ukládat všechny příkazy jazyka SQL u všech uživatelů do mezipaměti příkazů SQL, pokud jednotliví uživatelé tuto vlastnost nevypnou pomocí proměnné prostředí **STMT_CACHE** nebo příkazu SET STATEMENT CACHE OFF.

  ```
  onmode -e on
  ```

Následující tabulka shrnuje použití mezipaměti příkazů SQL u uživatelů v závislosti na nastavení konfiguračního parametru STMT_CACHE (nebo provedení příkazu **onmode -e**) a použití proměnné prostředí **STMT_CACHE** environment variable a příkazu SET STATEMENT CACHE.

Konfigurační parametr STMT_CACHE nebo příkaz onmode -e	Proměnná prostředí STMT_CACHE	Příkaz SET STATEMENT CACHE	Výsledné chování
0 (výchozí hodnota)	Nelze použít	Nelze použít	Mezipaměť pro příkazy není použita
1	0 (nebo nenastaveno)	OFF	Mezipaměť pro příkazy není použita
1	1	OFF	Mezipaměť pro příkazy není použita
1	0 (nebo nenastaveno)	ON	Mezipaměť pro příkazy je použita
1	1	ON	Mezipaměť pro příkazy je použita
1	1	Neproveden	Mezipaměť pro příkazy je použita
1	0	Neproveden	Mezipaměť pro příkazy není použita
2	1 (nebo nenastaveno)	ON	Mezipaměť pro příkazy je použita
2	1 (nebo nenastaveno)	OFF	Mezipaměť pro příkazy není použita
2	0	ON	Mezipaměť pro příkazy je použita
2	0	OFF	Mezipaměť pro příkazy není použita uživatelem
2	0	Neproveden	Mezipaměť pro příkazy není použita uživatelem
2	1 (nebo nenastaveno)	Neproveden	Mezipaměť pro příkazy je použita uživatelem

Umístění příkazů do mezipaměti paměti

Do mezipaměti příkazů SQL mohou být s výjimkami umístěny příkazy SELECT, UPDATE, INSERT a DELETE. Když databázový server kontroluje, zda se příkaz jazyka SQL nachází v mezipaměti, musí najít přesnou shodu.

Úplný seznam výjimek a úplný seznam požadavků na přesnou shodu naleznete v příručce *IBM Informix Guide to SQL: Syntax* v části týkající se příkazu SET STATEMENT CACHE.

Monitorování využití paměti u jednotlivých relací

Pokud chcete získat informace o paměti u jednotlivých relací, můžete použít argumenty volby **onstat -g**.

Postup identifikace příkazů jazyka SQL, které využívají velké množství paměti:

1. Pokud chcete zobrazit všechny uživatelské jednotkové procesy, použijte volbu **onstat -u**.
2. Pokud chcete zobrazit paměť všech relací a vidět, která relace využívá nejvíce paměti, použijte volbu **onstat -g ses**.
3. Pokud chcete zobrazit více podrobností o relaci, která využívá nejvíce paměti, použijte volbu **onstat -g ses** *session-id* .
4. Pokud chcete zobrazit paměť využívanou příkazy jazyka SQL, použijte volbu **onstat -g stm** *session-id* .

Příkaz onstat -g ses

Příkaz **onstat -g ses** využití paměti podle ID relací. Pokud relace sdílí paměťové struktury v mezipaměti příkazů SQL (SSC), hodnota ve sloupci **used memory** by měla být nižší, pokud je paměť vypnutá. Například Obrázek 13-6 uvádí příklad výstupu příkazu **onstat -g ses** v případě, že mezipaměť příkazů SQL není povolena a Obrázek 13-7 uvádí výstup poté, co je povolena a dotazy v relaci 4 jsou opětovně spuštěny. Obrázek 13-6 uvádí, že relace 4 využívá 45656 bajtů paměti. Obrázek 13-7 uvádí, že relace 4 využívá méně bajtů (36920), pokud je povolena mezipaměť příkazů SQL.

```
session                                        #RSAM    total      used
id       user     tty      pid      hostname  threads  memory     memory
12       informix -        0        -         0        12288      7632
4        informix 11       5158     smoke     1        53248      45656
3        informix -        0        -         0        12288      8872
2        informix -        0        -         0        12288      7632
```

Obrázek 13-6. Výstup příkazu onstat -g ses, pokud není mezipaměť příkazů SQL povolena

```
session                                        #RSAM    total      used
id       user     tty      pid      hostname  threads  memory     memory
17       informix -        0        -         0        12288      7632
16       informix 12       5258     smoke     1        40960      38784
4        informix 11       5158     smoke     1        53248      36920
3        informix -        0        -         0        12288      8872
2        informix -        0        -         0        12288      7632
```

Obrázek 13-7. Výstup příkazu onstat -g, pokud je mezipaměť příkazů SQL povolena

Obrázek 13-7 uvádí také paměť, která je přidělena a používána relací 16, která provádí stejné příkazy jazyka SQL jako relace 4. Relaci 16 je přiděleno méně celkové paměti (40960) a využívá méně paměti (38784) než relace 4 (Obrázek 13-6 uvádí hodnoty 53248 a 45656), protože využívá existující paměťové struktury mezipaměti příkazů SQL.

Příkaz onstat -g ses session-id

Volba **onstat -g ses** *session-id* zobrazí podrobné informace o relaci. Sloupce následujícího výstupu příkazu **onstat -g ses** *session-id* zobrazují využití paměti:

- Část výstupu Společné oblasti paměti
  - Sloupec **totalsize** zobrazuje počet aktuálně přidělených bajtů
  - Sloupec **freesize** zobrazuje počet nepřidělených bajtů
- Poslední řádek výstupu zobrazuje počet bajtů přidělených ze společné oblasti sscpool.

Obrázek 13-8 uvádí, že relaci 16 je aktuálně přiděleno 69632 bajtů, z nichž 11600 bajtů je přiděleno ze společné oblasti sscpool.

```
onstat -g ses 14


session                                        #RSAM    total      used
id       user     tty      pid      hostname  threads  memory     memory
14       virginia 7        28734    lyceum    1        69632      67384

tid      name     rstcb    flags    curstk   status
38       sqlexec  a3974d8  Y--P---  1656     cond wait(netnorm)

Memory pools    count 1
name         class addr      totalsize  freesize  #allocfrag #freefrag
14           V     a974020   69632      2248      156        2

...
Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
14    SELECT         vjp_stores         CR  Not Wait   0    0    9.03

Current statement name : slctcur

Current SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

Last parsed SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

 11600 byte(s) of memory is allocated from the sscpool
```

Obrázek 13-8. Výstup příkazu onstat -g ses session-id

Příkaz onstat -g sql session-id

Volba **onstat -g sql** *session-id* zobrazí informace o příkazech jazyka SQL, které relace provedla. Obrázek 13-9 uvádí, že výstup příkazu **onstat -g sql** *session-id* zobrazuje stejné informace jako dolní část výstupu příkazu **onstat -g ses** *session-id*, kterou uvádí Obrázek 13-8 a která zahrnuje počet bajtů přidělených ze společné oblasti sscpool.

```
onstat -g sql 14


Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
14    SELECT         vjp_stores         CR  Not Wait   0    0    9.03

Current statement name : slctcur

Current SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

Last parsed SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

 11600 byte(s) of memory is allocated from the sscpool
```

Obrázek 13-9. Výstup příkazu onstat -g sql session-id

Příkaz onstat -g stm session-id

Volba **onstat -g stm** *session-id* zobrazí informace o paměti, kterou využívá každý příkaz jazyka SQL relace. Obrázek 13-10 uvádí výstup příkazu **onstat -g stm** *session-id* pro stejnou relaci (14) jako u příkazu **onstat -g ses** *session-id*, který uvádí Obrázek 13-8 na stránce

13-32, a u příkazu **onstat -g sql** *session-id*, který uvádí Obrázek 13-9 na stránce 13-32. Pokud je mezipaměť příkazů SQL zapnuta, databázový server vytvoří haldy ve společném prostoru sscpool. Výstup příkazu **heapsz**, který uvádí Obrázek 13-10, zobrazuje, že tento příkaz jazyka SQL využívá 10056 bajtů, které jsou obsaženy ve společné oblasti sscpool zobrazené příkazem **onstat -g sql 14**.

```
onstat -g stm 14


session   14 ------------------------------------------------------------
 sdblock   heapsz   statement ('*' = Open cursor)
 aa11018   10056  *SELECT C.customer_num, O.order_num
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
```

Obrázek 13-10. Výstup příkazu onstat -g stm session-id

Monitorování využití mezipaměti příkazů SQL

Pokud zaznamenáte náhlé zvýšení doby odezvy dotazu, který využíval mezipaměť příkazů SQL, je možné, že byla vypuštěna nebo odstraněna položka. Databázový server vypustí položku z mezipaměti, pokud jeden z objektů, na kterých závisí dotaz, je změněn, takže u tohoto dotazu dojde ke zrušení platnosti položky rychlé vyrovnávací paměti datového slovníku. Následující operace způsobí selhání kontroly závislosti:

- Provedení libovolného příkazu jazyka pro definici dat (DDL) (například příkazu ALTER TABLE, DROP INDEX nebo CREATE INDEX), který může způsobit změnu plánu dotazů.
- Změna tabulky, která je spojena s jinou tabulkou pomocí referenčního omezení (v obou směrech).
- Provedení příkazu UPDATE STATISTICS FOR TABLE pro jednu tabulku nebo sloupec obsažený v dotazu.
- Přejmenování sloupce, databáze nebo indexu pomocí příkazu RENAME.

Pokud je položka označena jako vypuštěná nebo odstraněná, databázový server musí při příštím provádění příkazu jazyka SQL tento dotaz opětovně analyzovat a optimalizovat. Například Obrázek 13-11 na stránce 13-34 uvádí položky, které zobrazí příkaz **onstat -g ssc** po provedení příkazu UPDATE STATISTICS na tabulku**items** a **orders** mezi prováděním prvního a druhého příkazu jazyka SQL.

Část Statement Cache Entries: výstupu příkazu **onstat -g ssc**, který znázorňuje Obrázek 13-11, zobrazuje pole **flag**, které označuje, zda položka byla nebo nebyla odstraněna z mezipaměti příkazů SQL.

- U první položky se ve sloupci **flag** nachází hodnota DF, která označuje, že položka je zcela uložena v mezipaměti, ale nyní byla vypuštěna, protože byla zrušena platnost její položky.
- Druhá položka obsahuje stejný text příkazu jako třetí položka, který označuje, že položka byla během provedení po příkazu UPDATE STATISTICS opětovně analyzovýna a optimalizována .

```
onstat -g ssc

...
Statement Cache Entries:

lru hash ref_cnt hits  flag  heap_ptr  database     user
---------------------- ----- ---------------------------------------------------
...
  2  232       1    1 |  DF   aa3d020  vjp_stores   virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num

  3  232       1    0   -F  aa8b020  vjp_stores   virginia
|  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num
...
```

Obrázek 13-11. Výstup stejných příkazů onstat -g ssc pro vypuštěnou položku

Monitorování relací a jednotkových procesů

Při monitorování aktivity databázového serveru můžete zobrazovat množství aktivních relací a jednotkových procesů a množství zdrojů, které využívají. Monitorování relací a jednotkových procesů je důležité pro relace, které provádějí dotazy, stejně jako pro relace, které provádějí vkládání, aktualizace a odstraňování. Některé z informací, které lze monitorovat pro relace a jednotkové procesy, umožňují určit, zda aplikace nevyužívá příliš mnoho zdrojů.

Poznámka: Jednotkové procesy relací uložených procedur s nastaveními priotity PDQ a klauzulí GROUP BY nejsou uvolněny, dokud není relace dokončena.

Použití obslužných programů příkazového řádku

K monitorování relací a jednotkových procesů použijte následující volby obslužného programu **onstat**:

- **onstat -u**
- **onstat -g ath**
- **onstat -a act**
- **onstat -a ses**
- **onstat -g mem**
- **onstat -g stm**

onstat -u

Volba **onstat -u** zobrazuje informace aktivních jednotkových procesů, které vyžadují řídicí blok úloh databázového serveru. Aktivní jednotkové procesy zahrnují jednotkové procesy, které patří k uživatelským relacím, a také některé jednotkové procesy, které odpovídají procesům typu démon databázového serveru (například čištění stránek). Příklad výstupu tohoto programu uvádí část Obrázek 13-12 na stránce 13-35.

Pokud chcete určit, zda uživatel čeká na zdroj nebo používá příliš mnoho zámků, nebo získat představu o množství vstupů a výstupů, které uživatel provedl, použijte volbu **onstat -u**.

Výstup obslužného programu zobrazuje následující informace:

- Adresu každého jednotkového procesu
- Příznaky, které označují současný stav jednotkového procesu (například čekání na vyrovnávací paměť nebo čekání na kontrolní bod), zda se jedná o primární jednotkový proces relace a o jaký typ jednotkového procesu jde (například uživatelský jednotkový proces, jednotkový proces typu démon atd.)

  Informace o těchto příznacích naleznete v kapitole *IBM Informix Dynamic Server Administrator's Reference*.
- ID relace a přihlašovací ID uživatele pro relaci, ke které jednotkový proces patří

  ID relace s hodnotou 0 označuje, že se jedná o jednotkový proces typu démon.
- Zda jednotkový proces čeká na určitý zdroj a adresu tohoto zdroje
- Počet zámků, které jednotkový proces obsahuje
- Počet výzev ke čtení a počet výzev k zápisu, které jednotkový proces provedl
- Maximální počet aktuálně aktivních jednotkových procesů.

Pokud provádíte příkaz **onstat -u** v době, kdy databázový server provádí rychlou obnovu, může se na obrazovce zobrazit několik jednotkových procesů databázového serveru.

```
Userthreads
address  flags    sessid  user     tty     wait     tout locks nreads   nwrites
80eb8c   ---P--D 0        informix -        0        0    0     33       19
80ef18   ---P--F 0        informix -        0        0    0     0        0
80f2a4   ---P--B 3        informix -        0        0    0     0        0
80f630   ---P--D 0        informix -        0        0    0     0        0
80fd48   ---P--- 45       chrisw   ttyp3    0        0    1     573      237
810460   ------- 10       chrisw   ttyp2    0        0    1     1        0
810b78   ---PR-- 42       chrisw   ttyp3    0        0    1     595      243
810f04   Y------ 10       chrisw   ttyp2    beacf8   0    1     1        0
811290   ---P--- 47       chrisw   ttyp3    0        0    2     585      235
81161c   ---PR-- 46       chrisw   ttyp3    0        0    1     571      239
8119a8   Y------ 10       chrisw   ttyp2    a8a944   0    1     1        0
81244c   ---P--- 43       chrisw   ttyp3    0        0    2     588      230
8127d8   ----R-- 10       chrisw   ttyp2    0        0    1     1        0
812b64   ---P--- 10       chrisw   ttyp2    0        0    1     20       0
812ef0   ---PR-- 44       chrisw   ttyp3    0        0    1     587      227
 15 active, 20 total, 17 maximum concurrent
```

Obrázek 13-12. Výstup příkazu onstat -u

Příkaz onstat -g ath

Pokud chcete získat výpis všech jednotkových procesů, použijte volbu **onstat -g ath**. Oproti volbě **onstat -u**, tento výpis obsahuje vnitřní jednotkové procesy typu démon, které nemají řídicí blok úloh databázového serveru. Na druhé straně, zobrazení volby **onstat -g ath** neobsahuje ID relací (protože ne všechny jednotkové procesy patří k relacím).

Jednotkové procesy, které spustil primární jednotkový proces podpory rozhodování, mají název, který označuje jejich roli v dotazu pro podporu rozhodování. Například část Obrázek 13-13 uvádí čtyři jednotkové procesy, které patří k jednotkovému procesy podpory rozhodování.

```
Threads:
tid     tcb     rstcb   prty    status                  vp-class   name
...
11      994060  0       4       sleeping(Forever)         1cpu    kaio
12      994394  80f2a4  2       sleeping(secs: 51)        1cpu    btclean
26      99b11c  80f630  4       ready                     1cpu    onmode_mon
32      a9a294  812b64  2       ready                     1cpu    sqlexec
113     b72a7c  810b78  2       ready                     1cpu    sqlexec
114     b86c8c  81244c  2       cond wait(netnorm)        1cpu    sqlexec
115     b98a7c  812ef0  2       cond wait(netnorm)        1cpu    sqlexec
116     bb4a24  80fd48  2       cond wait(netnorm)        1cpu    sqlexec
117     bc6a24  81161c  2       cond wait(netnorm)        1cpu    sqlexec
118     bd8a24  811290  2       ready                     1cpu    sqlexec
119     beae88  810f04  2       cond wait(await_MC1)      1cpu    scan_1.0
120     a8ab48  8127d8  2       ready                     1cpu    scan_2.0
121     a96850  810460  2       ready                     1cpu    scan_2.1
122     ab6f30  8119a8  2       running                   1cpu    scan_2.2
```

Obrázek 13-13. Výstup příkazu onstat -g ath

Příkaz onstat -g act

Pokud chcete získat seznam aktivních jednotkových procesů, použijte volbu **onstat -g act**. Výstup příkazu **onstat -g act** zobrazuje část jednotkových procesů, které jsou uvedeny také ve výstupu příkazu **onstat -g ath**.

Příkaz onstat -g ses

Pokud chcete monitorovat zdroje přidělené relací uživateli, především relace spouštějící dotaz pro podporu rozhodování, použijte volbu **onstat -a ses** . Například relace číslo 49, kterou obsahuje Obrázek 13-14, spouští pět jednotkových procesů pro dotazy pro podporu rozhodování.

```
session                                         #RSAM    total      used
id       user     tty      pid      hostname threads  memory     memory
57       informix -        0     -        0        8192       5908
56       user_3   ttyp3    2318  host_10  1        65536      62404
55       user_3   ttyp3    2316  host_10  1        65536      62416
54       user_3   ttyp3    2320  host_10  1        65536      62416
53       user_3   ttyp3    2317  host_10  1        65536      62416
52       user_3   ttyp3    2319  host_10  1        65536      62416
51       user_3   ttyp3    2321  host_10  1        65536      62416
49       user_1   ttyp2    2308  host_10  5        188416     178936
2        informix -        0     -        0        8192       6780
1        informix -        0     -        0        8192       4796
```

Obrázek 13-14. Výstup příkazu onstat -g ses

Příkazy onstat -g mem a onstat -g stm

Pokud chcete získat informace o paměti využívané jednotlivými relacemi, použijte volby **onstat -g mem** a **onstat -g stm**. Relaci, na kterou se máte zaměřit, můžete určit pomocí sloupce **used memory** ve výstupu příkazu **onstat -g ses**.

Obrázek 13-15 zobrazuje ukázku výstupu příkazu **onstat -g ses** a výňatek z výstupů příkazů **onstat -g mem** a **onstat -g stm** pro relaci 16.

- Volba **onstat -g mem** zobrazuje celkové množství paměti využité jednotlivými relacemi.

  Sloupec **totalsize** výstupu příkazu **onstat -g mem 16** zobrazuje celkové množství paměti přidělené relaci.
- Volba **onstat -g stm** zobrazuje část celkové paměti přidělenou aktuálně připravenému příkazu SQL.

Sloupec **heapsz** výstupu volby **onstat -g stm 16**, který uvádí Obrázek 13-15, zobrazuje množství paměti přidělené aktuálně připravenému příkazu jazyka SQL.

```
Výstup příkazu onstat -g ses

session                                        #RSAM    total      used
id       user     tty      pid      hostname threads  memory     memory
18       informix -        0        -        0        12288      8928
17       informix 12       28826    lyceum   1        45056      33752
16       virginia 6        28743    lyceum   1        90112      79504
14       virginia 7        28734    lyceum   1        45056      33096
3        informix -        0        -        0        12288      10168
2        informix -        0        -        0        12288      8928



onstat -g mem 16

Pool Summary:
name         class addr     totalsize freesize #allocfrag #freefrag
16           V     a9ea020  90112     10608    159        5
...


onstat -g stm 16

session   16 ------------------------------------------------------------------
 sdblock  heapsz  statement ('*' = Open cursor)
 aa0d018   10056 *SELECT C.customer_num, O.order_num
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
```

Obrázek 13-15. Výstupy programu onstat, které určují paměť relace

Použití programu ON-Monitor k monitorování relací (operační systém UNIX)

Vyberte položku **User** z nabídky Status. Tato volba poskytne část informací, které zobrazuje obslužný program **onstat -u**. Zobrazí se následující informace:

- ID relace
- ID uživatele
- Počet zámků, které jednotkový proces obsahuje
- Počet výzev ke čtení a počet počet výzev k zápisu, které jednotkový proces provedl
- Příznaky, které označují současný stav jednotkového procesu (například čekání na vyrovnávací paměť nebo čekání na kontrolní bod), zda se jedná o primární jednotkový proces relace a o jaký typ jednotkového procesu jde (například uživatelský jednotkový proces, jednotkový proces typu démon atd.)

Ukázku výstupu uvádí Obrázek 13-16.

```
                        USER THREAD INFORMATION

                      Locks       Disk       Disk       User thread
Session   User        Held        Reads      Writes     Status

     0    informix       0          96          2       ------D
     0    informix       0           0          0       ------F
     0    informix       0           0          0       -------
    15    informix       0           0          0       Y-----M
     0    informix       0           0          0       ------D
    17    chrisw         1           3         34       Y------
```

Obrázek 13-16. Výstup volby User z nabídky Status programu ON-Monitor

Použití programu ISA k monitorování relací

Pokud chcete monitorovat uživatelské relace pomocí programu ISA, přejděte na stránku **Users** a klepněte na položku **Threads**. Program ISA využívá informace generované příkazem **onstat -u**. Klepnutím na tlačítko **Refresh** opětovně spustíte příkazy a zobrazíte aktuální informace.

Použití tabulek SMI

Následující informace získáte provedením dotazu na tabulku **syssessions**.

Sloupec	**Popis**
sid	ID relace
username	Uživatelské jméno (přihlašovací ID) uživatele
uid	ID uživatele
pid	ID procesu
connected	Čas spuštění relace
feprogram	Aplikace, která je spuštěna jako klient (program typu front-end)

Kromě toho některé sloupce obsahují příznaky, které označují, zda *primární* jednotkový proces relace čeká na zámek latch, zámek, protokol vyrovnávací paměti nebo transakci, zda se jedná o jednotkový proces programu ON–Monitor a zda se nachází v důležité části.

Důležité: Informace v tabulce **syssessions** jsou uspořádány podle relací a informace ve výstupu příkazu **onstat -u** jsou uspořádány podle jednotkových procesů. Narozdíl od výstupu příkazu **onstat -u** tabulka **syssessions** neobsahuje informace o jednotkových procesech typu démon.

Pokud chcete získat profil aktivity relace, proveďte dotaz na tabulku **syssesprof**. Tato tabulka obsahuje řádek pro každou relaci a sloupce, ve kterých jsou uloženy statistické údaje aktivity relací (například počet zámků, počet zápisů řádků, počet potvrzení, počet odstranění, atd.).

Úplný seznam sloupců tabulky **syssessions** a popis sloupců tabulky **syssesprof** naleznete v kapitole o databázi **sysmaster** v příručce *IBM Informix Dynamic Server Administrator's Reference*.

Monitorování transakcí

Monitorováním transakcí zaznamenáte otevřené transakce a zámky, které tyto transakce obsahují. Pomocí programu ISA lze monitorovat transakce a uživatelské relace. Program ISA využívá k zobrazování informací o relacích informace, které generují následující volby příkazového řádku obslužného programu **onstat**, jak uvádí následující tabulka. Klepnutím na tlačítko **Refresh** opětovně spustíte příkaz obslužného programu **onstat** a zobrazíte aktualizované informace.

Položky k monitorování	Výběr v programu ISA	Zobrazí výstup příkazu	Další informace
Statistické údaje transakcí	**Users > Transaction**	**onstat -x**	"Zobrazení transakcí pomocí příkazu onstat -x" na stránce 13-39
Statistické údaje o uživatelských relacích	**Users > Threads**	**onstat -u**	"Zobrazení uživatelských relací pomocí příkazu onstat -u" na stránce 13-41
Statistické údaje o zámcích	**Performance > Locks**	**onstat -k**	"Zobrazení zámků pomocí příkazu onstat -k" na stránce 13-40
Relace se spuštěnými příkazy SQL	**Users > Connections > session-id**	**onstat -g sql** *session-id*	"Zobrazení relací provádějících příkazy jazyka SQL" na stránce 13-42

Zobrazení transakcí pomocí příkazu onstat -x

Výstup příkazu **onstat -x** obsahuje následující informace o každé otevřené transakci:

- Adresu struktury transakce ve sdílené paměti
- Příznaky, které označují následující informace:
  - Aktuální stav transakce (uživatelské jednotkové procesy - připojené, pozastavené a čekající na odvolání)
  - Režim, ve kterém jsou transakce spuštěny (volně vázané nebo provázané)
  - Fázi, ve které se transakce nachází (fáze BEGIN WORK, připravena na potvrzení, potvrzovaná nebo potvrzená, odvolávaná)
  - Druh transakce (globální transakce, koordinátor, podřízený, koordinátor i podřízený)
- Jednotkový proces, který transakci vlastní
- Počet zámků, které transakce obsahuje
- Soubor logického protokolu, do kterého byl protokolován záznam BEGIN WORK
- Aktuální ID a pozici logického protokolu
- Úroveň izolace
- Počet pokusů o spuštění jednotkového procesu obnovení
- Koordinátora transakce (pokud transakci provádí podřízená transakce)
- Maximální počet souběžných transakcí od posledního spuštění databázového serveru

Tento obslužný program je zvláště užitečný pro monitorování globálních transakcí. Můžete například určit, zda má být transakce prováděna ve volně vázaném nebo v provázaném režimu. Tyto režimy transakcí mají následující charakteristiky:

- Volně vázaný režim

  Každá větev globální transakce má samostatný ID transakce (XID). Jedná se o výchozí režim.

  - Různé databázové servery koordinují transakce, ale nesdílejí zdroje. Žádné dvě transakce, i když přistupují ke stejné databázi, nemohou sdílet zámky.

- Záznamy ze všech větví globální transakce se v logickém protokolu zobrazují jako samostatné transakce.

- Provázaný režim

  Všechny větve globální transakce, které přistupují ke stejné databázi, sdílí stejný ID transakce (XID). Tento režim se vyskytuje jen ve správci transakcí Microsoft Transaction Server (MTS).

  - Různé databázové servery koordinují transakce a sdílejí zdroje, jako jsou zámky a záznamy protokolů. Větve se stejným ID sdílejí zámky a nemusí nikdy čekat na další větev se stejným XID, protože je v jeden moment aktivní pouze jedna větev.
  - Záznamy protokolů větví se stejným XID se zobrazují v logickém protokolu stejné transakce.

Obrázek 13-17 uvádí ukázku výstupu příkazu **onstat -x**. Poslední uvedená transakce je globální transakce, jak označuje hodnota G v páté pozici sloupce **flags**. Hodnota T ve druhé pozici sloupce **flags** označuje, že transakce je spuštěna v provázaném režimu.

```
Transactions
address  flags userthread locks  beginlg curlog  logposit   isol    retrys coord
ca0a018 A---- c9da018    0     0       5      0x18484c COMMIT  0
ca0a1e4 A---- c9da614    0     0       0      0x0      COMMIT  0
ca0a3b0 A---- c9dac10    0     0       0      0x0      COMMIT  0
ca0a57c A---- c9db20c    0     0       0      0x0      COMMIT  0
ca0a748 A---- c9db808    0     0       0      0x0      COMMIT  0
ca0a914 A---- c9dbe04    0     0       0      0x0      COMMIT  0
ca0aae0 A---- c9dcff8    1     0       0      0x0      COMMIT  0
ca0acac A---- c9dc9fc    1     0       0      0x0      COMMIT  0
ca0ae78 A---- c9dc400    1     0       0      0x0      COMMIT  0
ca0b044 AT--G c9dc9fc    0     0       0      0x0      COMMIT  0
10 active, 128 total, 10 maximum concurrent
```

Obrázek 13-17. Výstup příkazu onstat -x

Výstup, který uvádí Obrázek 13-17, zobrazuje, že tato větev transakce obsahuje 13 zámků. Pokud je transakce spuštěna v provázaném režimu, větve této transakce sdílejí zámky.

Zobrazení zámků pomocí příkazu onstat -k

Pokud chcete získat více podrobností o zámcích, které transakce obsahuje, použijte příkaz **onstat -k**. Chcete-li najít odpovídající zámky, porovnejte adresu ve sloupci **userthread** ve výstupu příkazu **onstat -x** s adresou ve sloupci **owner** výstupu příkazu **onstat -k**. Obrázek 13-18 uvádí ukázku výstupu příkazu **onstat -x** a odpovídajícího příkazu **onstat -k**. Hodnota a335898 ve sloupci **userthread** ve výstupu příkazu **onstat -x** odpovídá hodnotě ve sloupci **owner** dvou řádků výstupu příkazu **onstat -k**.

```
Výstup příkazu onstat -x

Transactions
address  flags userthread locks  beginlg curlog  logposit   isol    retrys coord
a366018  A---- a334018    0      0       1       0x22b048  COMMIT  0
a3661f8  A---- a334638    0      0       0       0x0       COMMIT  0
a3663d8  A---- a334c58    0      0       0       0x0       COMMIT  0
a3665b8  A---- a335278    0      0       0       0x0       COMMIT  0
a366798  A----  a335898     2      0       0       0x0       COMMIT  0
a366d38  A---- a336af8    0      0       0       0x0       COMMIT  0
 6 active, 128 total, 9 maximum concurrent

onstat -k

Locks
address  wtlist    owner     lklist   type     tblsnum  rowid    key#/bsiz
a09185c  0         a335898   0        HDR+S    100002   20a        0
a0918b0  0         a335898   a09185c  HDR+S    100002   204        0
 2 active, 2000 total, 2048 hash buckets, 0 lock table overflows
```

Obrázek 13-18. Výstup příkazů onstat -k a onstat -x

V příkladu, který uvádí Obrázek 13-18, uživatel vybírá řádek z tabulky. Uživatel používá dva zámky:

- Sdílený zámek na jedné databázi
- Sdílený zámek na druhé databázi

Zobrazení uživatelských relací pomocí příkazu onstat -u

ID relace můžete najít porovnáním adresy ve sloupci **userthread** výsledku příkazu **onstat -x** se sloupcem **address** výstupu příkazu **onstat -u**. Sloupec **sessid** na stejném řádku ve výstupu příkazu **onstat -u** poskytuje ID relace. Například Obrázek 13-19 zobrazuje adresu a335898 ve sloupci **userthread** výstupu příkazu **onstat -x**. Řádek výstupu příkazu **onstat -u** se stejnou adresou zobrazuje ID relace 15 ve sloupci **sessid**.

```
Výstup příkazu onstat -x

Transactions
address  flags userthread locks  beginlg curlog  logposit   isol    retrys coord
a366018  A---- a334018    0      0       1       0x22b048  COMMIT  0
a3661f8  A---- a334638    0      0       0       0x0       COMMIT  0
a3663d8  A---- a334c58    0      0       0       0x0       COMMIT  0
a3665b8  A---- a335278    0      0       0       0x0       COMMIT  0
a366798  A----  a335898     2      0       0       0x0       COMMIT  0
a366d38  A---- a336af8    0      0       0       0x0       COMMIT  0
 6 active, 128 total, 9 maximum concurrent


onstat -u

address  flags    sessid  user     tty     wait     tout locks nreads   nwrites
a334018  ---P--D 1        informix -        0        0    0     20       6
a334638  ---P--F 0        informix -        0        0    0     0        1
a334c58  ---P--- 5        informix -        0        0    0     0        0
a335278  ---P--B 6        informix -        0        0    0     0        0
 a335898   Y--P---  15        informix 1        a843d70  0    2     64       0
a336af8  ---P--D 11       informix -        0        0    0     0        0
 6 active, 128 total, 17 maximum concurrent
```

Obrázek 13-19. Získání ID relace pomocí sloupce userthread ve výstupu příkazu onstat -x

U transakce prováděné ve volně vázaném režimu může první pozice sloupce **flags** ve výstupu příkazu **onstat -u** zobrazovat hodnotu T. Tato hodnota T označuje, že jedna větev globální

transakce čeká na dokončení jiné větve. Tato situace by mohla nastat, pokud by se dvě různé větvě globální transakce (obě využívající stejnou databázi) pokusily pracovat současně se stejnou globální transakcí.

U transakce prováděné v provázaném režimu se tato hodnota T nezobrazuje, protože databázový server sdílí jednu strukturu transakce všem větvím, které přistupují ke stejné databázi v globální transakci. Najednou je připojena a aktivní pouze jedna větev. Tato větev nečeká na zámky, protože transakce vlastní všechny zámky používané různými větvemi.

Zobrazení relací provádějících příkazy jazyka SQL

Pokud chcete získat informace o příkazu jazyka SQL, který provedla každá relace, použijte příkaz **onstat -g sql** s odpovídajícím ID relace. Obrázek 13-20 uvádí ukázku výstupu této volby s použitím stejného ID relace získaného z ukázky **onstat -u**, kterou uvádí Obrázek 13-19.

```
onstat -g sql 15

Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
15    SELECT         vjp_stores         CR  Not Wait   0    0    9.03

Current statement name : slctcur

Current SQL statement :
  select l.customer_num, l.lname, l.company,   l.phone, r.call_dtime,
    r.call_descr from customer l, vjp_stores@gilroy:cust_calls r where
    l.customer_num = r.customer_num

Last parsed SQL statement :
  select l.customer_num, l.lname, l.company,   l.phone, r.call_dtime,
    r.call_descr from customer l, vjp_stores@gilroy:cust_calls r where
    l.customer_num = r.customer_num
```

Obrázek 13-20. Výstup příkazu onstat -g sql

Kapitola 14. Obslužný program onperf v systému UNIX

Obsah kapitoly

Kapitola popisuje obslužný program **onperf** - prostředí s okny, které lze použít ke sledování výkonu databázového serveru. Oblužný program **onperf** monitoruje databázový server spuštěný v operačním systému UNIX.

Obslužný program **onperf** umožňuje sledování většiny běžných parametrů databázové serveru, které jsou protokolovány obslužným programem**onstat**. Obslužný program **onperf** má však oproti obslužnému programu **onstat** zejména následující výhody:

- Hodnoty metriky zobrazuje graficky v reálném čase.
- Umožňuje zvolit metriku, která má být monitorována.
- Umožňuje posouvání zpět na předchozí hodnoty metriky a analyzovat trend.
- Umožňuje uložit údaje o výkonnosti do souboru pro pozdější prohlížení.

Přehled obslužného programu onperf

Tato část poskytuje přehled funkčnosti programu **onperf** a dalších nástrojů programu **onperf**.

Základní funkce programu onperf

Obslužný program **onperf** provádí následující základní funkce:

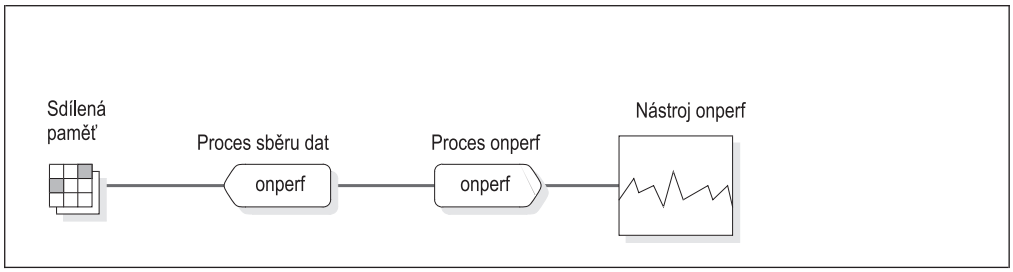
- V okně nástroje zobrazuje hodnoty metrik databázového serveru.
- Ukládá metriku databázového serveru do souboru.
- Dovoluje zkontrolovat hodnoty metriky databázového serveru, které jsou uložené v souboru.

Zobrazení hodnot metriky

Po spuštění programu **onperf** jsou aktivovány následující procesy:

- **Proces onperf.** Řídí zobrazení nástrojů **onperf**.
- **Proces kolektor dat.** Tento proces se připojuje ke sdílené paměti a předává informace o výkonnosti procesu **onperf**, který je zobrazuje v nástroji **onperf**.

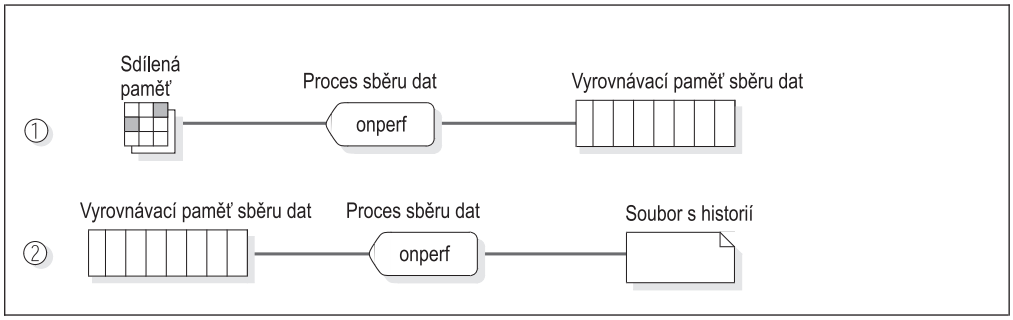
Nástroj **onperf** je oknem Motif, které je spravováno procesem **onperf** manages, jak je uvedeno na obrázku Obrázek 14-1.



Obrázek 14-1. Datový tok ze sdílené paměti do okna nástroje onperf

Ukládání hodnot metriky do souboru

Obslužný program **onperf** umožňuje nepřetržitě ukládat vybranou metriku do vyrovnávací paměti. Proces kolektor dat tyto metriky zapisuje do kruhové vyrovnávací paměti, která se nazývá *vyrovnávací paměť kolektoru dat*. Je-li vyrovnávací paměť zaplněna, kolektor dat pokračuje v přidávání dat a přepisuje tak nejstarší hodnoty. Obrázek 14-2 zobrazuje ukládání obsahu vyrovnávací paměti kolektoru dat do souboru.



Obrázek 14-2. Ukládání dat výkonnosti programem onperf

Obslužný program **onperf** může data v souboru historie ukládat v binárním nebo ASCII formátu. Binární formát je závislý na hostiteli, umožňuje však rychlý zápis dat. Formát ASCII je přenositelný mezi platformami.

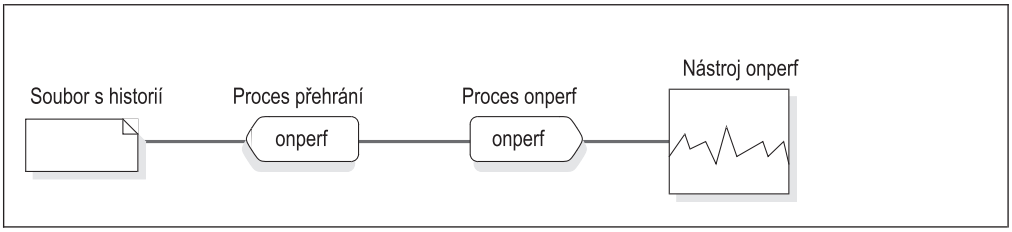
Lze zvolit sadu metrik, které jsou ukládány do vyrovnávací paměti kolektoru dat, a počet vzorků. Do vyrovnávací paměti lze ukládat všechny metriky. Tento způsob však může být paměťově náročný. Každé měření metriky vyžaduje 8 bajtů paměti. Pokud je například vzorkování prováděno frekvencí jednoho vzorku za sekundu, bude k uložení 200 metrik

o 3.600 vzorcích do vyrovnávací paměti zapotřebí přibližně 5,5 MB paměti. Pokud takový proces představuje nepřiměřeně velkou část paměti, je nutné snížit velikost vyrovnávací paměti, vzorkovací frekvenci nebo počet ukládaných metrik.

Konfiguraci velikosti vyrovnávací paměti nebo vzorkovací frekvence lze provést prostřednictvím dialogového okna Konfigurace. Další informace o dialogovém okně Konfigurace získáte v části “Nabídka Konfigurace nástroje Graf a dialogové okno Konfigurace” na stránce 14-8.

Prohlížení měření metrik

Obsah souboru historie si lze prohlédnout v okně nástrojů. Pokud má nástroj zobrazit soubor historie, program **onperf** spustí proces přehrávání, který čte data z disku a zasílá je do nástroje, který je zobrazuje. Proces přehrávání je podobný procesu kolektor dat, který je popsán v části “Ukládání hodnot metriky do souboru” na stránce 14-2. Místo čtení dat ze sdílené paměti však proces přehrávání čte data ze souboru historie. Obrázek 14-3 zobrazuje proces přehrávání.



Obrázek 14-3. Datový tok ze souboru historie do okna nástroje onperf

Nástroj onperf

Obslužný program **onperf** nabízí následující okna správce zobrazení Motif, která se nazývají *nástroje*, a která zobrazují hodnoty metriky:

- Nástroj Graf

  Tento nástroj umožňuje sledovat obecnou aktivitu výkonnosti. Nástroj lze použít k zobrazení libovolné kombinaci metrik, které program **onperf** podporuje, a k zobrazení obsahu souboru historie. Další informace naleznete v části “Nástroj Graf” na stránce 14-5.

- Nástroj Strom dotazů

  Tento nástroj zobrazuje průběh jednotlivých dotazů. Další informace naleznete v části “Nástroj Strom dotazů” na stránce 14-11.

- Nástroj Stav

  Nástroj zobrazuje informace o stavu databázového serveru a umožňuje ukládat do souboru data, která jsou obsažena ve vyrovnávací paměti kolektoru. Další informace naleznete v části “Nástroj Stav” na stránce 14-11.

- Nástroje Aktivita

  Tyto nástroje zobrazují specifické aktivity databázového serveru. Nástroje Aktivity zahrnují nástroje disku, relací, kapacity disku, fyzických procesorů a virtuálních procesorů. Nástroje fyzický procesor a virtuální procesor zobrazují informace o všech procesorech CPU resp. VP. Každý z ostatních nástrojů aktivity zobrazuje 10 nejvýše hodnocených instancí zdroje, hodnoceného dle vhodného měření aktivity. Další informace naleznete v části “Nástroje Aktivity” na stránce 14-12.

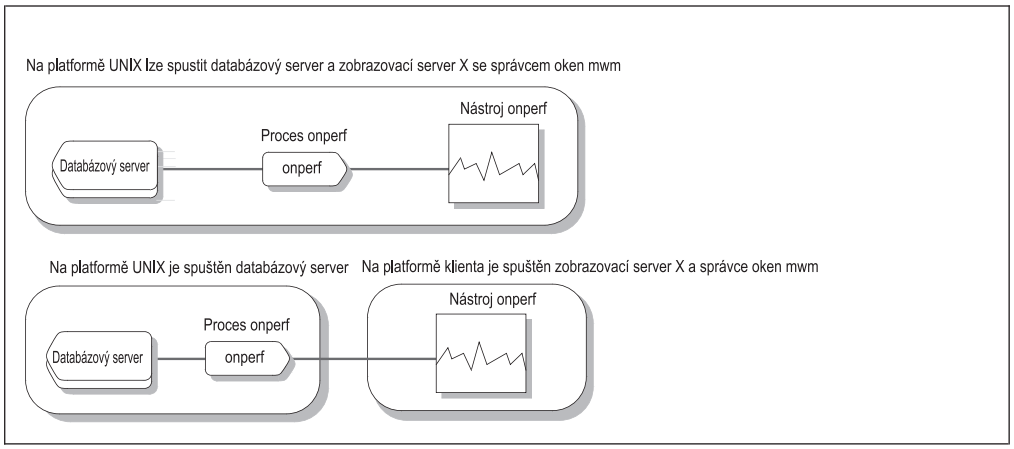
Požadavky na spuštění programu onperf

Po instalaci databázového serveru jsou následující spustitelné soubory zapsány do adresáře **$INFORMIXDIR/bin**:

- **onperf**
- **onedcu**
- **onedpu**
- **xtree**

Soubor nápovědy online onperf.hlp je navíc umístěn v adresáři **$INFORMIXDIR/hhelp**.

Po instalaci a spuštění databázového serveru v režimu online lze zobrazit nástroje programu **onperf** v počítači, ve kterém je spuštěn databázový server nebo ve vzdáleném počítači nebo terminálu, který je schopen komunikovat s instancí databázového serveru. Obrázek 14-4 zobrazuje obě možnosti. V obou případech musí být v počítači, ve kterém jsou spuštěny nástroje**onperf**, podporován terminál X-terminál okenního správce **mwm**.



Obrázek 14-4. Dvě možnosti spuštění programu onperf

Spuštění a ukončení nástroje onperf

Před spuštěním nástroje **onperf** nejprve nastavte následující proměnné prostředí na příslušné hodnoty:

- **DISPLAY**
- **LD_LIBRARY_PATH**

Proměnnou prostředí **DISPLAY** nastavte následovně:

```
Prostředí C shell         setenv DISPLAY displayname0:0 #

Prostředí Bourne shell    DISPLAY=displayname0:0 #
```

U těchto příkazů určuje položka *displayname* název počítače nebo terminálu serveru X-serveru, na kterém má být okno **onperf** zobrazeno.

Proměnnou prostředí **LD_LIBRARY_PATH** nastavte na vhodnou hodnotu tak, aby obsahovala cestu ke knihovnám správce oken Motif v počítači, ve kterém je nástroj **onperf** spuštěn.

Po správném nastavení proměnných prostředí můžete zobrazit okno grafického nástroje zadáním příkazu **onperf**, jak je popsáno v části “Uživatelské rozhraní nástroje onperf” na stránce 14-5.

Nástroj **onperf** ukončíte zvolením možnosti **Zavřít** u každého okna nástroje nebo pomocí možnosti **Ukončit** nástroje, nebo zvolením příkazu **Správce oken > Zavřít**.

Pokud vyvoláte nástroj **onperf** pro každý databázový server, můžete monitorovat více instancí databázového serveru v jednom okně správce Motif. Viz následující příkaz:

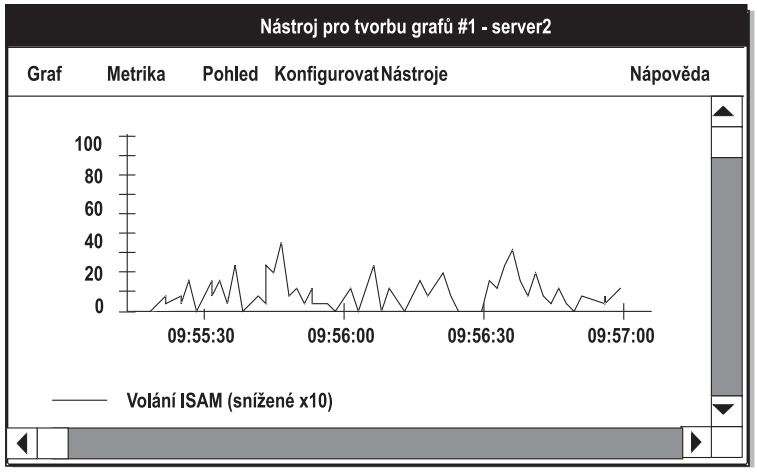
```
INFORMIXSERVER=instance1 ; export INFORMIXSERVER; onperf
INFORMIXSERVER=instance2 ; export INFORMIXSERVER; onperf
...
```

Uživatelské rozhraní nástroje onperf

Po spuštění nástroje **onperf** zobrazí obslužný program okno grafu nástroje. V okně grafu nástroje lze zobrazit další okna grafu nástroje spolu s nástrojem strom dotazů, kolektor dat a aktivity. Okna grafu nástroje nemají žádnou hierarchii. Lze je vytvářet a zavírat v libovolném pořadí.

Nástroj Graf

Nástroj graf je základním **onperf** rozhraním. Nástroj umožňuje zobrazit libovolnou sadu metrik databázového serveru, které nástroj kolektor dat **onperf** získá ze sdílené paměti. Obrázek 14-5 zobrazuje diagram nástroje graf.



Obrázek 14-5. Okno nástroje Graf

Okno nástroje graf nelze vyvolat z nástroje strom dotazů, nástroje stavu ani ze žádného z nástrojů aktivity.

Pruh titulku

Po spuštění nástroje **onperf**se zobrazí výchozí okno nástroje graf **NázevServeru** - databázový server, jehož název určuje proměnná prostředí **INFORMIXSERVER**, která je uvedena v pruhu titulků. Data jsou získávána ze sdílené paměti zobrazené instance databázového serveru.

Pokud konfigurace výchozího nástroje graf ještě nebyla uložena nebo načtena z disku, nástroj **onperf** nezobrazí v pruhu titulku název konfiguračního souboru.

Pokud je například v tomto okně nástroje graf otevřen soubor dat historie nazvaný **caselog.23April.2PM**, bude v pruhu titulku zobrazen text **caselog.23.April.23.April.2PM**.

Nabídka Graf nástroje Graf

Nabídka **Graf** poskytuje následující volby.

Volba	Použití
Nový	Vytvoří nový nástroj graf. Všechny nástroje grafu, které vytvoříte pomocí této volby, sdílejí stejné procesy kolektor dat a **onperf**. Nové nástroje graf vyvolávejte prostřednictvím této volby. Nespouštějte vícenásobné instance nástroje **onperf**.
Otevřít soubor historie	V nástroji graf načte dříve uložený soubor dat historie a umožní jeho prohlížení. Pokud takový soubor neexistuje, nástroj **onperf** zobrazí výzvu. Po vybrání souboru nástroj **onperf** spustí proces přehrávání a zobrazí soubor.
Uložit soubor historie	Uloží obsah vyrovnávací paměti kolektoru dat do souboru v binárním nebo ASCII formátu (podle nastavení v dialogovém okně Konfigurace).
Uložit soubor historie jako	Určuje název souboru, do kterého bude uložen obsah vyrovnávací paměť kolektoru dat.
Anotovat	Zobrazí dialogové okno, ve kterém je možné zadat popisku záhlaví a zápatí. Popisky jsou nepovinné. Popisky se zobrazí v grafu. Po uložení konfigurace grafu zahrne nástroj **onperf** tyto popisky do ukládaného souboru konfigurace.
Tisk	Zobrazí dialogové okno, které umožňuje vybrat cílový soubor. Obsah nástroje graf nelze přímo odeslat na tiskárnu. Nejprve je nutné prostřednictvím této volby vybrat soubor a následně odeslat soubor PostScript na tiskárnu.
Zavřít	Zavře nástroj. Pokud je nástroj posledním nástrojem relace **onperf**, chová se tato nabídka stejně jako volba **Ukončit**.
Ukončit	Ukončí nástroj **onperf**.

Důležité: Uložení aktuální konfigurace před načtením nové konfigurace provedete následovně. Nejprve vyberte příkaz **Konfigurace> Uložit konfiguraci** nebo **Konfigurace> Uložit konfiguraci jako**.

Nabídka Metriky nástroje Graf

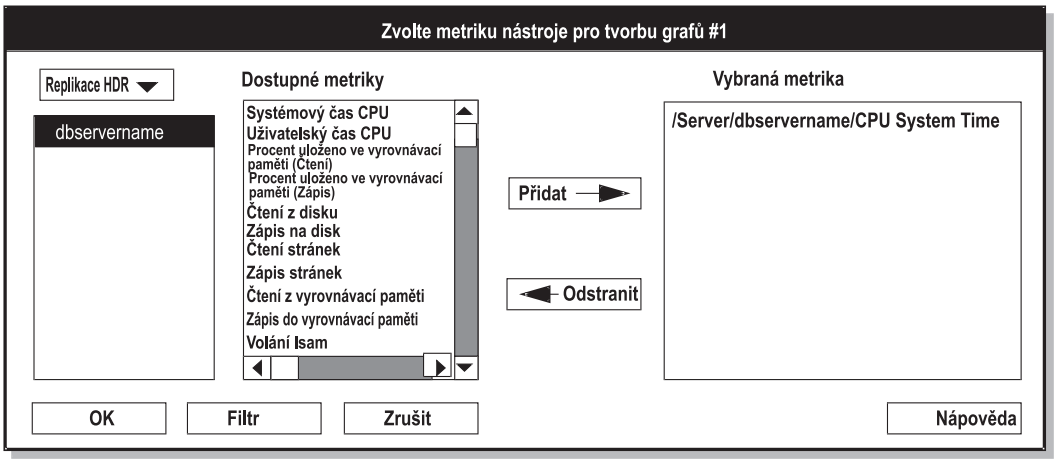
Třídy metrik zobrazovaných v nástroji graf lze zvolit v nabídce **Metriky**.

Metriky jsou členěny dle *třídy* a *rozsahu platnosti*. Po vybrání metriky, která má být zobrazena nástroji grafu, je nutné vybrat třídu, rozsah platnosti a název metriky.

Třída metriky je obecnou komponentou nebo aktivitou databázového serveru, kterou metrika monitoruje. *Rozsah platnosti metriky* je závislý na třídě metriky. Rozsah platnosti metriky někdy označuje určitou komponentu nebo aktivitu. V ostatních případech rozsah platnosti označuje veškeré aktivity určitého typu v rámci instance databázového serveru.

Nabídka **Metriky** obsahuje samostatné volby pro jednotlivé třídy metrik. Další informace o metrikách získáte v části “Způsob použití nástroje onperf” na stránce 14-12.

Po zvolení třídy, například **Server**, se zobrazí dialogové okno, podobné dialogovému oknu, které znázorňuje Obrázek 14-6.



Obrázek 14-6. Dialogové okno Vybrat metriku

Dialogové okno Vybrat metriku obsahuje tři rozbalovací seznamy. Rozbalovací seznam na levé straně obsahuje platné úrovně rozsahu platnosti pro vybranou třídu metriky. Pokud je například rozsah platnosti nastaven na hodnotu **Server**, zobrazí seznam název databázového serveru instance databázového serveru, která je právě monitorována. Pokud v seznamu vyberete rozsah platnosti, nástroj **onperf** zobrazí v seznamu uprostřed jednotlivé metriky, které jsou k dispozici v rámci zvoleného rozsahu platnosti. V seznamu lze zvolit více jednotlivých metrik. Do seznamu pro zobrazení je přidáte klepnutím na tlačítko **Přidat**. Pokud nemají být zobrazeny, klepněte na tlačítko **Odebrat**.

Tip: V jednom okně nástroje graf lze zobrazit více metrik jedné třídy. Lze například nejprve vybrat **ISAM volání**, **Otevření** a **Spuštění** třídy **Server**. Pokud ve stejném dialogovém okně zvolíte nabídku **Možnosti**, můžete vybrat jinou třídu metriky bez nutnosti uzavření stávajícího dialogového okna. Můžete například vybrat třídu metriky Bloky a přidat na displej metriky **Operace**, **Čtení** a **Zápisy**.

Tlačítko **Filtr** v dialogovém okně zobrazí další dialogové okno, ve kterém je možné filtrovat dlouhé řetězce textu, které jsou zobrazeny v dialogovém okně Metriky. V dialogovém okně Filtr lze také vybrat tabulky nebo fragmenty, pro které nejsou aktuálně zobrazeny metriky.

Nabídka Zobrazit nástroje Graf

Nabídka **Zobrazit** nabízí následující volby.

Volba	Použití
Čárový graf	Změna formátu nástroje graf na čárový formát. Čárový formát obsahuje vodorovné a svislé posuvníky. Svislý posuvník upravuje měřítko vodorovné časové osy. Při posunutí panelu vzhůru nástroj **onperf** měřítko zmenšuje a naopak. Vodorovný posuvník slouží k úpravě pohledu podél vodorovné časové osy. Barvu a šířku čáry v čárovém formátu změňte následovně. Klepněte na legendu nástroje graf. Nástroj **onperf** potom zobrazí dialogové okno Přizpůsobit metriku, které poskytuje volby barev a tloušťky čáry.
Vodorovný sloupcový graf	Změna formátu nástroje graf na vodorovný sloupcový formát.

Svislý sloupcový graf
: Změna formátu nástroje graf na svislý sloupcový formát.

Výsečový graf
: Změna formátu nástroje graf na výsečový formát. K zobrazení výsečového grafu je nutné vybrat alespoň dvě metriky.

Rychlá změna měřítka os
: Změní měřítko os podle nejvyšší hodnoty, která je aktuálně zobrazena. Tlačítko vypíná funkci automatické změny měřítka.

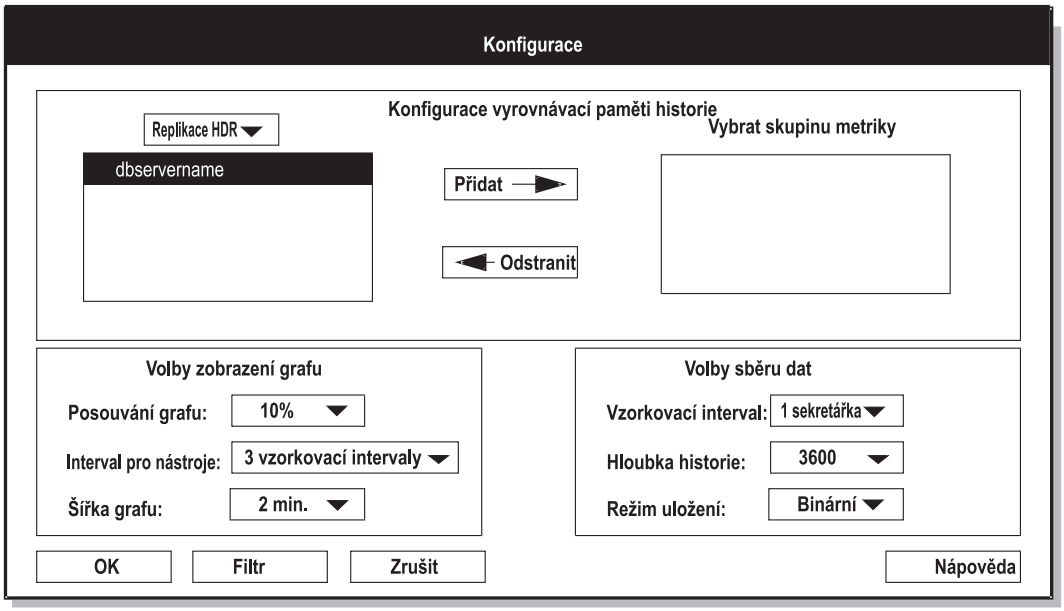
Konfigurace os
: Zobrazí dialogové okno Konfigurace os. Dialogové okno slouží k nastavení pevné hodnoty rozsahu svislé osy nebo k nastavení automatické změny měřítka.

Nabídka Konfigurace nástroje Graf a dialogové okno Konfigurace

Nabídka **Konfigurace** nabízí následující volby.

Volba	Použití
Upravit konfiguraci	Zobrazí dialogové okno Konfigurace, které dovoluje změnit nastavení aktuální vyrovnávací paměti kolektoru dat, možnosti zobrazení nástroje graf a možnosti kolektoru dat. Dialogové okno Konfigurace znázorňuje Obrázek 14-7 na stránce 14-9.
Otevřít konfiguraci	Znovu spustí nástroj **onperf** s nastaveními, která jsou uložena v konfiguračním souboru. Neuložená data vyrovnávací paměti kolektoru dat budou ztracena. Pokud není zadán žádný konfigurační soubor a výchozí konfigurační soubor neexistuje, zobrazí se následující chybová zpráva: `Otevření souboru` *`název_souboru`* `se nezdařilo.` Pokud zadaný konfigurační soubor neexistuje, nástroj **onperf** zobrazí výzvu.
Uložit konfiguraci	Ukládá aktuální konfiguraci do souboru. Pokud není určen žádný konfigurační soubor, nástroj **onperf** zobrazí výzvu.
Uložit konfiguraci jako	Ukládá konfigurační soubor. Volba vždy zobrazí výzvu k zadání názvu souboru.

Konfiguraci možností kolektoru dat, zobrazení nástroje graf a metrik, jejichž data mají být shromažďovány, provedete následovně. Zobrazte dialogové okno Konfigurace pomocí volby **Upravit konfiguraci**.



Obrázek 14-7. Dialogové okno Konfigurace

Dialogové okno Konfigurace nabízí následující volby zobrazení.

Volba	Použití
Konfigurace historie vyrovnávací paměti	Dovoluje vybrat třídu a rozsah platnosti metriky, která má být zahrnuta do vyrovnávací paměti kolektoru dat. Nástroj Kolektor dat shromažďuje informace o všech metrikách, které patří do vybrané třídy a rozsahu platnosti.
Možnosti zobrazení nástroje Graf	Dovoluje upravit velikost části grafu, která se posouvá vlevo v případě, že zobrazení dosáhne pravého okraje, výchozí časové rozpětí grafu a frekvenci aktualizace zobrazení.
Možnosti nástroje Kolektor dat	Řídí proces sběru dat. Interval vzorkování označuje časový interval mezi zaznamenávanými vzorky. Délka historie určuje počet vzorků, které jsou zachovávány ve vyrovnávací paměti. Režim ukládání určuje, zda mají být data kolektoru dat ukládána v binárním nebo ASCII formátu.

Nabídka Nástroje nástroje Graf

Ostatní nástroje lze vyvolat prostřednictvím nabídky **Nástroje**. Nabídka nabízí následující volby.

Volba	Použití
Strom dotazů	Spustí nástroj Strom dotazů. Další informace naleznete v části “Nástroj Strom dotazů” na stránce 14-11.
Stav	Spustí nástroj Stav. Další informace naleznete v části “Nástroj Stav” na stránce 14-11.
Aktivita disku	Spustí nástroj Aktivita disku. Další informace naleznete v části “Nástroje Aktivity” na stránce 14-12.
Aktivita relace	Spustí nástroj Aktivita relace. Další informace naleznete v části “Nástroje Aktivity” na stránce 14-12.

Kapacita disku
: Spustí nástroj Kapacita disku. Další informace naleznete v části "Nástroje Aktivity" na stránce 14-12.

Aktivita fyzického procesoru
: Spustí nástroj Aktivita fyzického procesoru. Další informace naleznete v části "Nástroje Aktivity" na stránce 14-12.

Aktivita virtuálního procesoru
: Spustí nástroj Aktivita virtuálního procesoru. Další informace naleznete v části "Nástroje Aktivity" na stránce 14-12.

Změna měřítka metriky

Nástroj **onperf** při zobrazení metriky automaticky upravuje měřítko svislé osy tak, aby pojmula nejvyšší hodnotu. Pomocí dialogového okna jej lze ustanovit pro aktuální zobrazení. Další informace naleznete v části "Nabídka Zobrazit nástroje Graf" na stránce 14-7.

Zobrazení hodnot nedávné historie

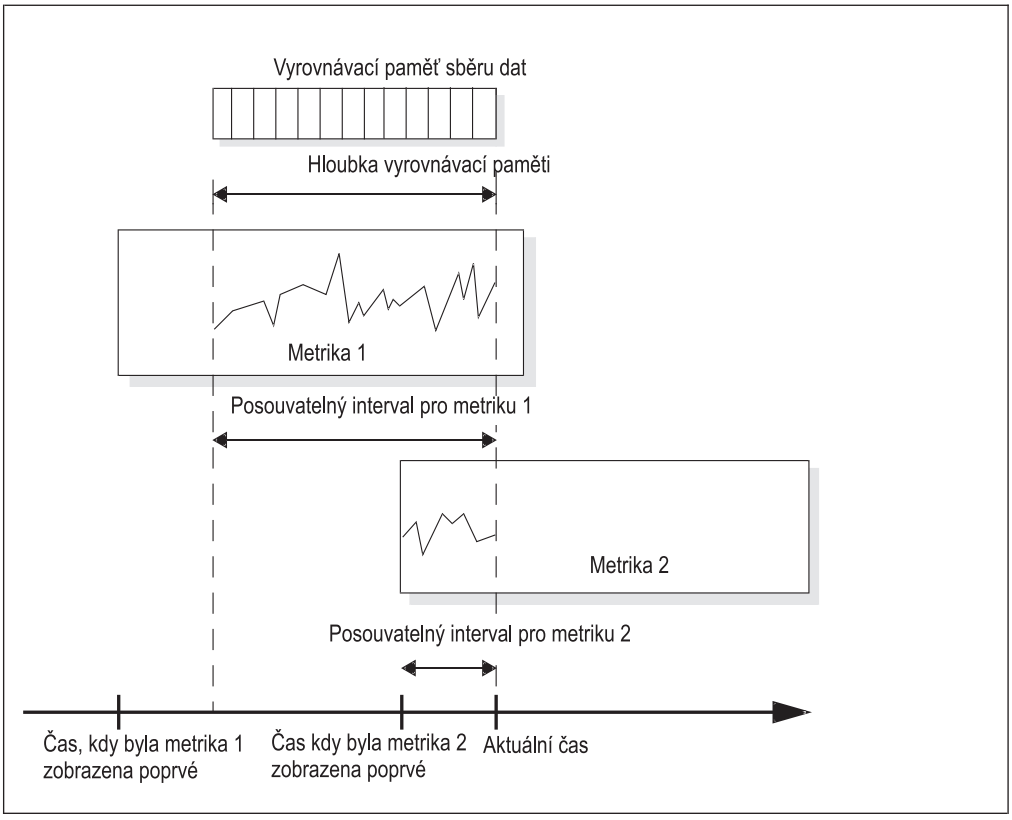
Obslužný program **onperf** umožňuje posouvat předchozími hodnotami metriky, které jsou zobrazené v čárovém grafu. Je tak možno analyzovat nedávný trend. Časový interval, o který je možno se posunout zpět, je *menší* z následujících intervalů:

- Časový interval, po který již je metrika zobrazena.
- Interval historie, který je určen v dialogovém okně Konfigurace nástroje graf.

  Časový úsek, po kterém lze posouvat zpět, nemůže překročit hloubku vyrovnávací paměti kolektoru dat.

  Další informace naleznete v části "Nabídka Konfigurace nástroje Graf a dialogové okno Konfigurace" na stránce 14-8.

Obrázek 14-8 uvádí maximální velikost intervalů, o které lze posouvat, pro metriky s různým časovým rozpětím.



Obrázek 14-8. Maximální intervaly, o které lze posouvat, pro metriky s různým časovým rozpětím.

Nástroj Strom dotazů

Nástroj strom dotazů umožňuje monitorovat výkon jednotlivých dotazů. Jedná se o samostatný spustitelný nástroj, který nevyužívá proces kolektor dat. Nástroj strom dotazů nelze uložit do souboru.

Nástroj obsahuje tlačítka **Vybrat relaci** a **Konec**. Pokud vyberte relaci, která vykonává dotaz, okno s detaily zobrazí operátory jazyka SQL plánu vykonávání dotazu. Nástroj strom dotazů reprezentuje jednotlivé operátory jazyka SQL pomocí políček. Každé políčko obsahuje číselník, který uvádí rychlost v řádkách za sekundu a číslo, které označuje vstupní řádky. V některých případech nemusí být možné uvést v okně detailů všechny operátory jazyka SQL. Menší okno zobrazuje operátory jazyka SQL jako malé ikony.

Tlačítko **Ukončit** slouží k ukončení nástroje strom dotazů.

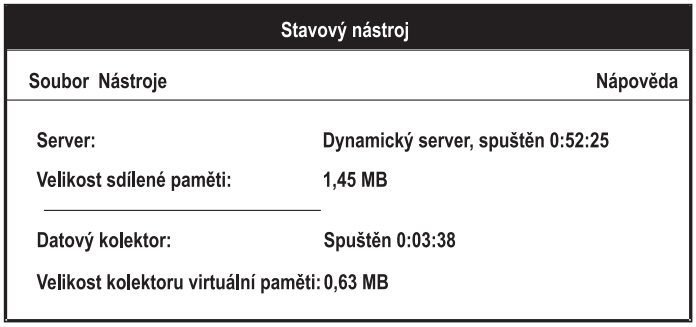
Nástroj Stav

Nástroj stavu dovoluje vybrat metriky, které mají být ukládány kolektorem dat. Tento nástroj lze také použít pro uložení dat, která jsou aktuálně obsažena ve vyrovnávací paměti kolektoru dat, do souboru. Obrázek 14-9 zobrazuje nástroj Stav.

Nástroj stav zobrazuje:

- Dobu, po kterou je nástroj kolektor dat spuštěn.
- Velikost oblasti procesu kolektoru dat, zvanou *velikost virtuální paměti kolektoru*

Pokud vyberete jinou metriku pro ukládání ve vyrovnávací paměti kolektoru dat, zobrazí se různé hodnoty virtuální paměti kolektoru.



Obrázek 14-9. Okno nástroje Stav

V nástroji stav poskytuje nabídka **Soubor** následující volby.

Volba	Použití
Zavřít	Zavře nástroj. Pokud je nástroj posledním nástrojem relace nástroje **onperf**, chová se tato nabídka stejně jako volba Ukončit.
Ukončit	Tato volba ukončí nástroj **onperf**.

Nástroje Aktivity

Nástroje aktivity jsou speciálním případem nástroje Graf, které zobrazují instance specifické aktivity v závislosti na ohodnocení aktivity vhodnou metrikou. Lze zvolit z následujících nástrojů aktivity:

- Nástroj aktivity disku, který zobrazuje horních 10 aktivit, měřeno počtem celkových operací.
- Nástroj aktivity relace, který zobrazuje horních deset aktivit, měřeno počtem volání ISAM plus PDQ za sekundu.
- Nástroj kapacity disku, který zobrazuje horních 10 aktivit, měřeno volným místem v MB.
- Nástroj aktivity fyzického procesoru zobrazuje všechny procesory, které nejsou ve stavu nečinnosti.
- Nástroj aktivity virtuálního procesoru zobrazuje všechny virtuální procesory, hodnocené součtem času uživatele a systému virtuálního procesoru.

Nástroje aktivity používají sloupcový formát grafu. Měřítko nástroje aktivity nelze změnit ručně. Nástroj **onperf** tuto hodnotu vždy nastavuje automaticky.

Nabídka **Graf** nabízí možnosti zavření, tisku a ukončení nástroje aktivity.

Způsob použití nástroje onperf

Následující část popisuje různé způsoby použití obslužného programu **onperf**.

Běžné monitorování

Nástroj **onperf** lze používat k usnadnění běžného monitorování. Můžete například zobrazit několik metrik v okně nástroje graf a spouštět tento nástroj v průběhu dne. Zobrazením těchto metrik můžete kdykoliv monitorovat výkonnost databázového serveru.

Diagnostika náhlých poklesů výkonnosti

Pokud dojde k náhlému poklesu výkonnosti, můžete prohlédnout historii hodnot metrik databázového serveru a posoudit vývoj. Nástroj **onperf** dovoluje posouvat v časovém intervalu zpět, jak je popsáno v části “Zobrazení hodnot nedávné historie” na stránce 14-10.

Diagnostika poklesu výkonnosti

Potíže s výkonností, které se vyvíjejí postupně, se obtížně odhalují. Pokud například dojde ke zvýšení doby odezvy databázového serveru, nemusí být z pohledu na aktuální metriky zřejmé, která hodnota je za zpomalení zodpovědná. Pokles výkonnosti může být tak pozvolný, že není možné odhalit změnu v nedávné historii hodnot metriky. Nástroj **onperf** dovoluje provádět srovnání v delších intervalech - dovoluje ukládat hodnoty metriky do souboru. Více informací naleznete v části “Nástroj Stav” na stránce 14-11.

Metriky nástroje onperf

Následující část popisuje různé třídy metrik. Jednotlivé části uvádějí dostupné úrovně rozsahu platnosti a popisují jednotlivé metriky v rámci třídy.

Výkonnost databázového serveru závisí na mnoha faktorech, včetně konfigurace operačního systému, databázového serveru a zatížení. Popsat vztah mezi metrikami nástroje **onperf** a specifickými parametry výkonu je velice obtížné.

Zde jsou popisovány jednotlivé metriky bez spekulace nad specifickými problémy s výkonem, které mohou označovat. Experimentálně lze zjistit, které metriky nejlépe monitorují výkonnost specifické instance databázového serveru.

Metriky databázového serveru

Rozsahem platnosti těchto metrik je vždy pojmenovaný databázový server, který označuje databázový server jako celek, nikoliv komponentu databázového serveru nebo diskového prostoru.

Název metriky	Popis
Čas CPU systému	Čas systému definovaný dodavatelem platformy.
Čas CPU uživatele	Čas uživatele definovaný dodavatelem platformy.
Procentní část mezipaměti (Čtení)	Procentní část všech operací čtení, které jsou čteny z mezipamětibez nutnosti čtení z disku. Použit je následující výpočet: `100 * ((čtení_z_vyrovnávací_paměti – čtení_z_disku) / (čtení_z_vyrovnávací_paměti))`
Procentní část mezipaměti (Zápis)	Procentní část všech operací zápisu, které jsou zápisem do vyrovnávací paměti. Použit je následující výpočet: `100 * ((zápisy_do_vyrovnávací_paměti – zápisy_na_disk) / (zápisy_do_vyrovnávací_paměti))`
Čtení z disku	Celkový počet operací čtení z disku.
Zápis na disk	Celkový počet operací zápisu na disk.
Čtení stránky	Počet operací čtení stránky z disku.
Zápis stránky	Počet stránek přenesených na disk.
Čtení z vyrovnávací paměti	Počet čtení z mezipaměti.
Zápisy do vyrovnávací paměti	Počet zápisů do mezipaměti.
Volání	Počet volání obdržených databázovým serverem.

Název metriky	Popis
Čtení	Počet volání operace čtení obdržených databázovým serverem.
Zápisy	Počet volání operace zápisu obdržených databázovým serverem.
Opětovné zápisy	Počet volání operace opětovného zápisu obdržených databázovým serverem.
Odstranění	Počet volání operace odstranění obdržených databázovým serverem.
Potvrzení transakce	Počet volání operace potvrzení transakce obdržených databázovým serverem.
Odvolání transakce	Počet volání operace odvolání transakce obdržených databázovým serverem.
Přetečení tabulky	Počet situací, kdy byla tabulka tblspace nedostupná (přetečení).
Přetečení zámku	Počet situací, kdy byla tabulka zámků nedostupná (přetečení).
Přetečení uživatele	Počet situací, kdy byla tabulka user nedostupná (přetečení).
Kontrolní body	Počet zapsaných kontrolních bodů od začátku spuštění sdílené paměti databázového serveru
Čekání vyrovnávací paměti	Počet čekání jednotkových procesů na přístup k vyrovnávací paměti.
Čekání na uzamknutí	Počet čekání jednotkových procesů na přístup k zámku.
Požadavky uzamknutí	Počet požadavků jednotkových procesů na uzamknutí.
Zablokování	Počet zjištěných zablokování.
Prodlevy zablokování	Počet vypršení prodlevy zablokování (včetně distribuovaných transakcí).
Čekání na kontrolní body	Počet čekání na kontrolní body, tj. počet jednotkových procesů, které čekají na dokončení kontrolního bodu.
Dopředné čtení stránek indexovaných dat	Počet operací dopředného čtení klíčů indexu.
Listy indexu Dopředné čtení	Počet operací dopředného čtení pro uzly listu indexu.
Pouze datová cesta Dopředné čtení	Počet operací dopředného čtení stránek dat.
Požadavky zachytávání	Počet požadavků zachytávání.
Čtení ze sítě	Počet přečtených zpráv ASF.
Zápisy na síť	Počet zapsaných zpráv ASF.
Přidělená paměť	Velikost virtuálního adresního prostoru databázového serveru v kilobajtech.
Použitá paměť	Velikost sdílené paměti databázového serveru v kilobajtech.
Využití dočasného prostoru	Velikost sdílené paměti vyhrazené pro dočasné tabulky, v kilobajtech.
Volání PDQ	Celkový počet paralelně zpracovaných akcí, které databázový server provedl.
Paměť DSS	Velikost paměti aktuálně použitá pro dotazy pro podporu rozhodování.

Metriky bloků disku

Metriky bloků disku používají jako rozsah platnosti metriky název cesty bloku.

Název metriky	Popis
Operace disku	Celkový počet operací vstupu-výstupu vztahujících se k označenému bloku.
Čtení z disku	Celkový počet operací čtení z bloku.
Zápis na disk	Celkový počet operací zápisu do bloku.
Volné místo (MB)	Velikost dostupného volného místa v megabajtech.

Metriky otáček disku

Metriky otáček disku používají jako rozsah platnosti metriky název cesty diskového zařízení nebo souboru operačního systému.

Název metriky	Popis
Operace disku	Celkový počet operací vstupu-výstupu vztahujících se k označenému disku nebo souboru operačního systému s vyrovnávací pamětí.
Čtení z disku	Celkový počet operací čtení z disku nebo souboru operačního systému.
Zápis na disk	Celkový počet operací zápisů na disk nebo do souboru operačního systému.
Volné místo	Velikost dostupného volného místa v megabajtech.

Metriky fyzického procesoru

Metriky fyzického procesoru používají jako rozsah platnosti identifikátor fyzického procesoru (například0 nebo 1) nebo **Celkem**.

Název metriky	Popis
Procentní část času CPU systému	Čas CPU systému fyzických procesorů.
Procentní části času CPU uživatele	Čas CPU uživatele fyzických procesorů.
Procentní část nečinnosti CPU	Čas nečinnosti CPU fyzických procesorů.
Procentní část času CPU	Součet času CPU systému a uživatele fyzických procesorů.

Metriky virtuálního procesoru

Tyto metriky používají jako rozsah platnosti třídu virtuálního procesoru (cpu, aio, kaio, atd.). Každá hodnota metriky představuje součet přes všechny instance této třídy virtuálního procesoru.

Název metriky	Popis
Čas uživatele	Souhrnný čas uživatele pro danou třídu.
Čas systému	Souhrnný čas systému pro danou třídu.
Operace semaforu	Celkový počet operací semaforu.
Zaneprázdněná čekání	Počet situací, kdy virtuální procesor třídy zabránil přepnutí kontextu prováděním smyčky před uspáním.
Spiny	Počet průchodů smyčkou.
V/V operace	Počet operací vstupu-výstupu za sekundu.
V/V čtení	Počet operací čtení za sekundu.
V/V zápis	Počet operací zápisu za sekundu.

Metriky relace

Tyto metriky používají jako rozsah platnosti aktivní relaci.

Název metriky	Popis
Čtení stránky	Počet operací čtení stránky z disku v rámci relace.
Zápis stránky	Počet operací zápisu stránky na disk v rámci relace.
Počet jednotkových procesů	Počet jednotkových procesů spuštěných v rámci relace.
Požadavky uzamknutí	Počet požadavků uzamknutí vyvolaných relací.
Čekání na uzamknutí	Počet operací čekání na uzamknutí jednotkových procesů relace.
Zablokování	Počet zablokování jednotkových procesů, patřících relaci.
Prodlevy zablokování	Počet prodlev zablokování jednotkových procesů, patřících relaci.
Záznamy protokolu	Počet záznamů protokolu zapsaných relací.
Volání ISAM	Počet operací volání ISAM relace.
Čtení ISAM	Počet operací čtení ISAM relace.
Zápis ISAM	Počet operací zápisu ISAM relace.
Opakovaný zápis ISAM	Počet operací opakovaného zápisu ISAM relace.
Odstranění ISAM	Počet operací odstranění ISAM relace.
Potvrzení transakce ISAM	Počet operací potvrzení transakcí ISAM relace.
Odvolání transakce ISAM	Počet operací odvolání transakcí ISAM relace.
Dlouhé transakce	Počet dlouhých transakcí vlastněných relací.
Čtení z vyrovnávací paměti	Počet operací čtení z vyrovnávací paměti provedených relací.
Zápisy do vyrovnávací paměti	Počet operací zápisu do vyrovnávací paměti provedených relací.
Použité místo žurnálu	Velikost použitého prostoru logického žurnálu.
Maximální použité místo žurnálu	Značka maxima použitého místa logického žurnálu pro tuto relaci.
Sekvenční prohledávání	Počet sekvenčních prohledávání zahájených relací.
Volání PDQ	Počet paralelně zpracovávaných akcí pro dotazy, vyvolané relací.
Přidělená paměť	Paměť přidělená pro relaci, v kilobajtech.
Použitá paměť	Paměť využitá relací, v kilobajtech.

Metriky prostoru Tblspace

Tyto metriky používají jako rozsah platnosti název prostoru tblspace. Název prostoru tblspace je složen z názvu databáze, dvojtečky a názvu tabulky (*database:table*). V případě fragmentovaných tabulek reprezentuje prostor tblspace součet všech fragmentů v tabulce. Měření jednotlivých fragmentů fragmentované tabulky lze získat prostřednictvím metrik fragmentace.

Název metriky	Popis
Požadavky uzamknutí	Celkový počet požadavků na uzamknutí prostoru tblspace.
Čekání na uzamknutí	Počet čekání jednotkových procesů na získání zámku pro prostor tblspace.
Zablokování	Počet zablokování, vztahujících se k prostoru tblspace.
Prodlevy zablokování	Počet prodlev zablokování, vztahujících se k prostoru tblspace.
Čtení	Počet operací čtení prostoru tblspace.
Zápisy	Počet operací zápisu prostoru tblspace.
Opětovné zápisy	Počet operací opětovného zápisu prostoru tblspace.
Odstranění	Počet operací odstranění prostoru tblspace.
Volání	Celkový počet operací volání prostoru tblspace.
Čtení z vyrovnávací paměti	Počet operací čtení z vyrovnávací paměti prostoru tblspace.
Zápisy do vyrovnávací paměti	Počet operací zápisu do vyrovnávací paměti prostoru tblspace.
Sekvenční prohledávání	Počet sekvenčních prohledávání prostoru tblspace.
Procentní část volného místa	Procentní část volné části prostoru tblspace.
Přidělené stránky	Počet stránek přidělených prostoru tblspace.
Použité stránky	Počet stránek přidělených prostoru tblspace, které byly zapsány.
Stránky dat	Počet stránek přidělených prostoru tblspace, které byly použity jako stránky dat.

Metriky fragmentace

Tyto metriky používají jako rozsah platnosti prostor dbspace jednotlivého fragmentu tabulky.

Název metriky	Popis
Požadavky uzamknutí	Celkový počet požadavků na uzamknutí fragmentu.
Čekání na uzamknutí	Počet čekání jednotkových procesů na získání zámku fragmentu.
Zablokování	Počet zablokování, vztahujících se k fragmentu.
Prodlevy zablokování	Počet prodlev zablokování, vztahujících se fragmentu.
Čtení	Počet operací čtení fragmentu.
Zápisy	Počet operací zápisu fragmentu.
Opětovné zápisy	Počet operací opětovného zápisu fragmentu.
Odstranění	Počet operací odstranění fragmentu.
Volání	Celkový počet operací volání fragmentu.
Čtení z vyrovnávací paměti	Počet operací čtení z vyrovnávací paměti fragmentu.
Zápisy do vyrovnávací paměti	Počet operací zápisu do vyrovnávací paměti fragmentu.
Sekvenční prohledávání	Počet sekvenčních prohledávání fragmentu.
Procentní část volného místa	Procentní část volné části fragmentu.
Přidělené stránky	Počet stránek přidělených fragmentu.
Použité stránky	Počet stránek přidělených fragmentu, do kterých bylo zapisováno.
Stránky dat	Počet stránek přidělených fragmentu, které byly použity jako stránky dat.

Dodatek A. Případové studie a příklady

Tento dodatek obsahuje případovou studii a několik rozšířených příkladů ladění výkonu, které jsou popsány v této příručce.

Případová studie

Následující případová studie znázorňuje situaci, ve které jsou disky přetíženy. Zobrazuje kroky, pomocí kterých lze izolovat příznaky a identifikovat problém na základě počáteční zprávy od uživatele, a potřebnou nápravu.

U databázové aplikace, u které nebylo dosaženo požadované propustnosti, bude pomocí kontroly zjištěno, jak je možné zvýšit výkon. Nástroje pro sledování operačního systému zjistí, že nebyla využita velká část času, protože se čekalo na vstup - výstup. Administrátor databázového serveru zvýšením počtu virtuálních procesorů umožní řídit souběžný vstup - výstup více procesory. Propustnost se však nezvýší, což znamená, že došlo k přetížení jednoho nebo více disků.

Chcete-li ověřit nečinnost vstupu - výstupu, musí administrátor databázového serveru identifikovat přetížené disky a prostory dbspace, které se na těchto discích nacházejí.

Postup identifikování přetížených disků a prostorů dbspace nacházejících se na těchto discích:

1. Pomocí volby **onstat -g ioq** zkontrolujete asynchronní vstupy - výstupy. Obrázek A-1 zobrazuje výstup.

```
AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
opt   0      0      0        0        0        0        0
msc   0      0      0        0        0        0        0
aio   0      0      0        0        0        0        0
pio   0      0      1        1        0        1        0
lio   0      0      1      341        0      341        0
gfd   3      0      1      225        2      223        0
gfd   4      0      1      225        2      223        0
gfd   5      0      1      225        2      223        0
gfd   6      0      1      225        2      223        0
gfd   7      0      0        0        0        0        0
gfd   8      0      0        0        0        0        0
gfd   9      0      0        0        0        0        0
gfd  10      0      0        0        0        0        0
gfd  11      0     28    32693    29603     3090        0
gfd  12      0     18    32557    29373     3184        0
gfd  13      0     22    20446    18496     1950        0
```

Obrázek A-1. Výstup volby onstat -g ioq

Obrázek A-1 znázorňuje důležité výsledky ve sloupcích **maxlen** a **totalops**:

- Sloupec **maxlen** zobrazuje nejobjemnější nevyřízenou událost požadavků vstupu - výstupu nashromážděných ve frontě. Poslední tři fronty jsou mnohem objemnější než jakákoliv jiná fronta v seznamu.
- Sloupec **totalops** zobrazuje stokrát více operací vstupu - výstupu dokončených během posledních tří front než u jakékoliv jiné fronty v seznamu.

Sloupce **maxlen** a **totalops** ukazují, že je zátěž vstupu - výstupu kriticky nevyvážená.

Dalším způsobem, jak zkontrolovat aktivitu vstupu - výstupu, je použití volby **onstat -g iov**. Pomocí této volby zobrazíte méně detailní přehled virtuálních procesorů.

2. Aktivitu AIO každého diskového zařízení zkontrolujete pomocí volby **onstat -g iof** (viz Obrázek A-2).

```
AIO global files:
gfd pathname         totalops  dskread dskwrite  io/s
3 /dev/infx2              0        0        0   0.0
4 /dev/infx2              0        0        0   0.0
5 /dev/infx2              2        2        0   0.0
6 /dev/infx2            223        0      223   0.5
7 /dev/infx4              0        0        0   0.0
8 /dev/infx4              1        0        1   0.0
9 /dev/infx4            341        0      341   0.7
10 /dev/infx4             0        0        0   0.0
11  /dev/infx5           32692    29602     3090   67.1
12  /dev/infx6           32556    29372     3184   66.9
13  /dev/infx7           20446    18496     1950   42.0
```

Obrázek A-2. Výstup volby onstat -g iof

Tento výstup ukazuje diskové zařízení přidružené ke každé frontě. V závislosti na uspořádání bloků může být několik front přidruženo ke stejnému zařízení. V tomto případě je celková aktivita pro **/dev/infx2** součtem sloupce **totalops** u front 3, 4, 5 a 6, což je 225 operací. Tato aktivita je ve srovnání s **/dev/infx5**, **/dev/infx6** a **/dev/infx7** stále nedůležitá.

3. Chcete-li určit prostory dbspace, které budou zahrnuty do zátěže vstupu - výstupu, použijte volbu **onstat -d,** (viz část Obrázek A-3.

```
Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
c009ad00 1        1        1        1        N        informix rootdbs
c009ad44 2        2001     2        1        N T      informix tmp1dbs
c009ad88 3        1        3        1        N        informix oltpdbs
c009adcc 4        1        4        1        N        informix histdbs
c009ae10 5        2001     5        1        N T      informix tmp2dbs
c009ae54 6        1        6        1        N        informix physdbs
c009ae98 7        1        7        1        N        informix logidbs
c009aedc 8        1        8        1        N        informix runsdbs
c009af20  9         1        9        3        N        informix  acctdbs
 9 active, 32 total

Chunks
address  chk/dbs offset   size     free    bpages   flags pathname
c0099574 1   1   500000   10000    9100             PO-   /dev/infx2
c009960c 2   2   510000   10000    9947             PO-   /dev/infx2
c00996a4 3   3   520000   10000    9472             PO-   /dev/infx2
c009973c 4   4   530000   250000   242492           PO-   /dev/infx2
c00997d4 5   5   500000   10000    9947             PO-   /dev/infx4
c009986c 6   6   510000   10000    2792             PO-   /dev/infx4
c0099904 7   7   520000   25000    11992            PO-   /dev/infx4
c009999c 8   8   545000   10000    9536             PO-   /dev/infx4
c0099a34 9    9    250000  450000    4947              PO-    /dev/infx5
c0099acc 10   9    250000  450000    4997              PO-    /dev/infx6
c0099b64 11   9    250000  450000    169997            PO-    /dev/infx7
 11 active, 32 total
```

Obrázek A-3. Zobrazení volby onstat -d

Ve výstupu **Chunks** ukazuje sloupec **pathname** diskové zařízení. Sloupec **chk/dbs** ukazuje počet bloků a prostorů dbspace, které se v každém disku nacházejí. V tomto případě je v každém z přetížených disků definovaný pouze jeden blok. Každý blok je přidružený k prostoru dbspace č. 9.

Výstup **Dbspaces** zobrazuje název prostoru dbspace, který je přidružený ke každému jeho číslu. V tomto případě jsou všechny tři přetížené disky součástí prostoru dbspace **acctdbs**.

Ačkoliv původní konfigurace disku přidělila prostoru dbspace **acctdbs** tyto tři disky, aktivita v rámci tohoto prostoru naznačuje, že se nejedná o dostatečné množství. Protože je zátěž mezi těmito třemi disky v podstatě rovnoměrná, nezdá se, že by tabulky byly chybně uspořádané nebo nesprávně fragmentované. Vyššího výkonu však můžete dosáhnout přidáním fragmentů z jiných disků do jedné nebo více velkých tabulek v tomto prostoru dbspace nebo přesunutím některých tabulek do jiných disků s menším zatížením.

Dodatek B. Usnadnění

Cílem IBM je poskytovat produkty pro každého, bez ohledu na věk či zdravotní handicap.

Funkce usnadňující přístup v rámci produktu IBM Informix Dynamic Server

Funkce usnadňující přístup pomáhají uživateli s tělesným postižením, například s postižením pohybového ústrojí či se zrakovým postižením, úspěšně využívat produkty informačních technologií.

Funkce usnadnění přístupu

Následující seznam obsahuje hlavní funkce systému IBM Informix Dynamic Server. Tyto funkce podporují:

- Operace prováděné pouze pomocí klávesnice.
- Rozhraní, která obvykle používají čtečky obrazovky.
- Přílohy alternativního vstupu a výstupu zařízení.

Tip: Informační centrum systému IBM Informix Dynamic Server a související příručky usnadňují přístup pro čtečky IBM Home Page Reader. Všechny funkce můžete ovládat pomocí klávesnice, nemusíte používat myš.

Navigace pomocí klávesnice

Tento produkt používá standardní Microsoft Windows navigační klávesy.

Informace související s usnadněním přístupu

IBM se zavázala vytvářet dokumentaci přístupnou postiženým osobám. Naše příručky jsou k dispozici ve formátu HTML, umožňují tedy přístup pomocí pomocných technologií, jako je například software pro čtení obrazovky. Diagramy syntaxe jsou v našich příručkách k dispozici ve formátu desítkových čísel oddělených tečkami.

Příručky pro systém IBM Informix Dynamic Server můžete prohlížet ve formátu PDF (Adobe Portable Document Format) pomocí aplikace Adobe Acrobat Reader.

IBM a usnadnění přístupu

Další informace o závazcích, které má IBM k usnadnění přístupu, naleznete v *Centru IBM pro usnadnění přístupu* na webu http://www.ibm.com/able.

Poznámky

IBM nemusí nabízet produkty, služby nebo vlastnosti zmiňované v tomto dokumentu ve všech zemích. Informace o produktech a službách, které jsou momentálně ve vaší zemi dostupné, můžete získat od zástupce společnosti IBM pro vaši oblast. Žádný z odkazů na produkty, programové vybavení nebo služby není zamýšlen jako tvrzení, že lze použít pouze tyto produkty, programové vybavení nebo služby společnosti IBM. Jako náhrada mohou být použity libovolné funkčně ekvivalentní produkty, programové vybavení nebo služby, které neporušují žádné intelektuální vlastnické právo společnosti IBM. Za operace prováděné produkty, programy nebo službami, které nepochází od společnosti IBM, nese zodpovědnost uživatel.

Společnost IBM může mít patenty nebo podané žádosti o patent, které zahrnují předmět tohoto dokumentu. Vlastnictví tohoto dokumentu vám nedává žádná práva k těmto patentům. Písemné žádosti o licenci můžete posílat na adresu:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pokud máte zájem o licenci v zemi s dvoubajtovou znakovou sadou (DBCS), kontaktujte zastoupení společnosti IBM ve vaší zemi nebo písemně zastoupení společnosti IBM na adrese:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

Následující odstavec se netýká Velké Británie ani kterékoliv jiné země, kde taková opatření odporují místním zákonům: SPOLEČNOST INTERNATIONAL BUSINESS MACHINES CORPORATION IBM POSKYTUJE TUTO PUBLIKACI TAK, JAK JE, BEZ ZÁRUKY JAKÉHOKOLIV DRUHU (VYJÁDŘENÉ VÝSLOVNĚ ČI VYPLÝVAJÍCÍ Z OKOLNOSTÍ) VČETNĚ, A TO ZEJMÉNA, JAKÝCHKOLIV ZÁRUK PRODEJNOSTI A VHODNOSTI PRO URČITÝ ÚČEL A ZÁRUK NEPORUŠENÍ PRÁV TŘETÍCH STRAN VYPLÝVAJÍCÍCH Z OKOLNOSTÍ. Toto prohlášení se na vás nemusí vztahovat, pokud váš stát nedovoluje zřeknutí se výslovné a předpokládané záruky v některých transakcích.

Tato příručka může obsahovat technické nepřesnosti nebo typografické chyby. Informace zde uvedené jsou pravidelně aktualizovány a v příštích vydáních této příručky již budou tyto změny zahrnuty. IBM může kdykoliv bez upozornění zdokonalovat nebo měnit produkty a programy popsané na těchto stránkách.

Jakékoliv odkazy v těchto informacích na webové stránky jiných společností než IBM jsou poskytovány pouze pro větší pohodlí uživatele a nemohou být žádným způsobem vykládány jako schválení těchto webových stránek společností IBM. Materiály obsažené na takových webových stránkách nejsou součástí materiálů tohoto produktu společnosti IBM a mohou být používány pouze na vlastní riziko.

IBM může, pokud to považuje za vhodné, používat nebo distribuovat libovolné informace, které jí poskytnete, aniž by tím vznikl jakýkoliv závazek IBM vůči vám.

Držitelé licence tohoto programu, kteří si přejí mít přístup i k informacím o tomto programu za účelem (i) výměny informací mezi nezávisle vytvořenými programy a jinými programy (včetně tohoto) a (ii) vzájemného použití sdílených informací, mohou kontaktovat:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Informace tohoto typu mohou být za odpovídajících podmínek dostupné. V některých případech může být požadováno zaplacení poplatku.

Program popsaný v tomto dokumentu a všechny materiály s ním související, které podléhají licenci, jsou dodávány IBM v souladu s textem smlouvy IBM Customer Agreement, IBM International Program License Agreement nebo smlouvy ekvivalentní, uzavřené mezi zákazníkem a IBM.

Všechny zde uvedené informace o výkonu byly zjištěny v řízeném prostředí. Výsledky zjištěné v jiném provozním prostředí se tudíž mohou výrazně lišit. Některá měření byla provedena v systémech s vývojovým prostředím a neexistuje žádná záruka, že tato měření budou stejná v obecně dostupných systémech. Některá měření mohla být také odhadnuta extrapolací. Skutečné výsledky se mohou lišit. Uživatelé tohoto dokumentu by měli příslušné údaje ověřit ve vlastním prostředí.

Informace týkající se produktů jiných společností byly získány od dodavatelů těchto produktů, z jejich tištěných materiálů nebo z jiných veřejně dostupných zdrojů. IBM netestovala tyto produkty a nemůže potvrdit spolehlivost jejich provozu, kompatibilitu nebo jiné tvrzení týkající se těchto produktů. Otázky týkající se možností produktů jiných společností by měly být adresovány dodavatelům těchto produktů.

Všechna tvrzení o budoucím zaměření nebo úmyslech IBM mohou být bez upozornění změněna nebo zrušena a představují pouze hrubý nástin cílů a podmínek společnosti.

Všechny uvedené ceny jsou současnými cenami pro koncové zákazníky navrženými IBM a mohou být bez upozornění změněny. Ceny prodejců se mohou od těchto cen lišit.

V rámci informací zde uvedených jsou uvedeny příklady sestav a dat, které jsou používány při každodenních operacích. Kvůli úplnosti obsahují příklady jmen jednotlivců, společností, značek a produktů. Všechna tato jména jsou smyšlená a jakákoli podobnost s existujícími jmény či adresami je čistě náhodná.

LICENCE NA AUTORSKÁ PRÁVA:

Tyto informace obsahují ukázkové programy ve zdrojovém jazyce, které ilustrují programovací techniky na různých platformách. Tyto vzorové programy můžete kopírovat, měnit a distribuovat v libovolné formě bez nutnosti úhrady IBM, za účelem vývoje, používání, marketingu nebo distribuce aplikačních programů přizpůsobených aplikačnímu programovému rozhraní pro operační platformu, pro kterou byly vzorové programové napsány. Tyto příklady nebyly důkladně testovány za veškerých podmínek. IBM proto nemůže zaručit spolehlivost, možnost opravy či funkčnost těchto programů ani tyto záruky nelze nijak vyvozovat. Tyto vzorové programy můžete kopírovat, měnit a distribuovat v libovolné formě bez úhrady IBM, za účelem vývoje, použití, marketingu nebo distribuce aplikačních programů odpovídajícím aplikačnímu programovému rozhraní IBM.

Každá kopie nebo část těchto vzorových programů nebo jakákoli odvozená práce musí zahrnovat následující doložku o autorských právech:

© (název vaší společnosti) (rok). Části tohoto kódu jsou odvozeny od kódu vzorových programů IBM Corp. Vzorové programy. © Copyright IBM Corp. (zadejte rok nebo roky). Všechna práva vyhrazena.

Pokud si prohlížíte tyto informace v souboru, nemusí se fotografie a barevné ilustrace zobrazit.

Ochranné známky

Následující termíny se používají v příručkách k produktu IBM Informix a jsou zároveň ochrannými známkami společnosti International Business Machines Corporation ve USA a případně v dalších jiných zemích:

AIX	MQSeries
C-ISAM	NUMA-Q
Cloudscape	OS/2
DataBlade	OS/390
DB2	OS/400
DB2 Connect	RedBack
DB2 Universal Database	Red Brick
Distributed Relational Database Architecture	RETAIN
Dynamic Connect	SystemBuilder
IBM	UniData
Informix	UniVerse

Adobe, Acrobat, Portable Document Format (PDF) a PostScript jsou ochranné známky nebo registrované ochranné známky Adobe Systems Incorporated v USA a případně v dalších jiných zemích.

Intel, logo Intel, Intel Inside, logo Intel Inside, Intel Centrino, logo Intel Centrino, Celeron, Intel Xeon, Intel SpeedStep, Itanium a Pentium jsou ochranné známky nebo registrivcané ochranné známky společnosti Intel Corporation či jejích dceřiných společností v USA a případně v dalších jiných zemích.

Java a všechny ochranné známky a loga související s jazykem Java jsou ochranné známky Sun Microsystems, Inc. v USA a případně v dalších jiných zemích.

Microsoft, Windows, Windows NT a logo Windows jsou ochranné známky Microsoft Corporation v USA a případně v dalších jiných zemích.

UNIX je registrovaná ochranná známka The Open Group v USA a případně v dalších jiných zemích.

Linux je registrovaná ochranná známka Linuse Torvaldse v USA a případně v dalších jiných zemích.

Další názvy společností, produktů nebo služeb mohou být ochranné známky nebo servisní známky jiných stran.

Rejstřík

Speciální znaky

Čísla

A

B

C

Č

D

E

F

G

H

CH

I

J

K

L

M

N

O

P

Q

R

Ř

S

T

U

V

W

Z

Vytištěno v Dánsku společností IBM Danmark A/S.

G229-1393-00

Spine information:

IBM Informix **Verze 11.1**

Řízení výkonu systému IBM Informix Dynamic Server