Optimalizace

Ing. Marek Sušický, RNDr. Ondřej Zýka

Obsah

- Úvod
- Indexy
- Optimalizátor
- Joiny
- Bulk operace
- Peklo jménem ORM (Object-relational mapping)

1 Úvod

Optimalizace

- Dvě pravidla optimalizace:
  - pravidlo č. 1: Nedělejte ji.
  - pravidlo č. 2: Zatím ji nedělejte.
- Tím není myšleno že nemáte optimalizovat, ale abyste nedělali „premature optimization“ (předčasnou optimalizaci), v jejímž jméně se páchají různá zvěrstva bez skutečné potřeby (řešící neexistující problém)
- Primárně programujte (vytvořte DB schéma, napište PL/SQL kód) tak aby dával logický smysl a aby vracel správné výsledky
- Používejte standardní principy softwarového vývoje. Odzkoušené patterny, neprogramujte to, co je již hotovo, pište čitelný a srozumitelný kód.
- Až pokud máte odzkoušeno že v daném procesu je výkonnostní problém, teprve pak přistupte k optimalizaci a řešte ho

Optimalizace

- Ladění procesu se bude velmi lišit podle toho co vlastně máte výkonem na mysli
- **propustnost** – cílem je maximální rychlost zpracování všech dat
- **latence** – cílem je minimální latence jednotlivých požadavků
- **škálovatelnost** – cílem je proces napsat tak aby dobře fungoval i při navýšení množství vstupních dat / současně pracujících uživatelů apod.
- často se zapomíná že neplatí rovnost mezi vysokou propustností a nízkou latencí (při vysoké propustnosti se často stává že latence zpracování některých požadavků dosahuje neakceptovaných hodnot)
- v podstatě záleží na tom jestli ladíte interaktivní nebo batch proces, batch procesy se ladí na celkovou propustnost, interaktivní na nízkou latenci

Brzdy výkonu

- Pomalé SQL dotazy (tradičně)
- Algoritmické brzdy
- Spousta maličkých dotazů (join na úrovni aplikace, dotahování detailů)
- Zbytečné změny dat

Základní doporučení

- Načítejte data co nejvíce zpracovaná
- Nechte na databázi (určitě to dokáže lépe)
  - Joiny tabulek
  - Výběr dat
  - Grupování a statistické výpočty
- Provádějte minimum zápisů
- Nechte na databázi řešení konkurenčního přístupu k datům (určitě to dokáže lépe)
- Provádějte změny najednou (co nejkratší transakce)
- Šetřete úpravy indexů, zápisy do undo/redo

Načítejte data co nejvíce zpracovaná

- Rozdrobení na maličké selecty je často důsledkem „naivního“ přepisu business procesu do PL/SQL podoby, např.
  - načtěte informace o transakci
  - načtěte informace o pobočce na které byla transakce zadána
  - načtěte informace o zákazníkovi na jehož účtu byla transakce provedena
  - může být realizováno pomocí jednoho SQL dotazu a nebo pomocí tří SQL
- každý SQL příkaz má určitou režii – a to i když se jedná o soft-parse (o hard-parse ani nemluvě. tam je to ještě horší)
- databáze má joinování poladěné, dokáže ho optimalizovat pomocí vhodného algoritmu (v závislosti na množství dat zvolí nested loop, merge join, hash join apod.)
- jednotlivé SQL často sklouznou na hromadu index scanů, které jsou poměrně „nákladné“ kvůli random I/O operacím, a ve výsledku je „ruční“ join výrazně horší než skutečný join

Příklad

- **špatně**: načítání dat pomocí malých selectů

```
 FOR t IN (SELECT * FROM transakce) LOOP
   FOR u IN (SELECT * FROM ucet WHERE id = t.ucet_id)
LOOP
      -- nejake zpracovani
   END LOOP;
 END LOOP;
```

- **dobře**: načítání dat v jednom joinu

```
 FOR x IN (SELECT * FROM transakce t JOIN ucet u
                      ON (t.ucet_id = u.id)) LOOP
    -- nejake zpracovani
 END LOOP;
```

Příklad

- **špatně**: zpracování batche s update pro každý záznam

```
FOR t IN (SELECT ucet_id, vyse FROM transakce) LOOP
   UPDATE ucet SET aktualni_stav = aktualni_stav + vyse
             WHERE id = t.ucet_id;
END LOOP;
```

- **dobře**: konsolidace zápisů

```
FOR t IN (SELECT ucet_id, SUM(vyse) AS vyse FROM transakce
                                    GROUP BY ucet_id)
LOOP
   UPDATE ucet SET aktualni_stav = aktualni_stav + vyse
             WHERE id = t.ucet_id;
END LOOP;
```

- 2

Indexy

Typy indexů

Typ indexu	Použití
B-tree index	<, =, > (interval hodnot)
Hash index	Existence
Bitmap index	Počet hodnot, prvek množiny
GIST index	Umožňuje definovat vlastní vyhledávací predikátory (pro multidimenzionální kostky …)
Fulltext index	Textové zpracování

- Lokální a globální indexy pro tabulka s partitions
- Struktury pro in-memory databáze
- Speciální datové struktury (grafové databáze)

B-tree index – příklad

kořen

klíč řádek blok

Blok 1001			
Bennet	Chet	1421,1	1007
Karsen	Kit	1876,4	1306
Smith	Ade	1242,3	1062

vnitřní blok indexu

Blok 1007			
Bennet	Chet	1421,1	1132
Fox	John	1317,3	1133
Hunter	Leon	1213,1	1127

Blok 1306			
Karsen	Kit	1876,4	1198
Larn	Pard	1451,2	1199
Peters	Mary	1856,4	1200

listová úroveň indexu

Blok 1132		
Bennet	Chet	1421,1
Burns	Saly	1409,4
Claim	Dave	1129,3
Dull	Rob	1409,1

Blok 1133		
Fox	John	1317,3
Greane	David	1876,4
Green	Mitch	1213,2
Greene	Joe	1409,2

Blok 1127		
Hunter	Leon	1213,1
Jetkins	Paul	1212,2

data

Blok 1212			
Larry	John	254	A3
Jetkins	Paul	244	C3
White	Susan	156	A1

Blok 1213			
Hunter	Leon	124	A3
Green	Mitch	125	B1
Smith	Ade	156	A3

Blok 1421			
Bennet	Chet	101	B2
Ringer	John	144	C1

Blok 1409			
Dull	Rob	128	B1
Greene	Joe	142	A2
Port	Joe	156	C3
Burns	Saly	128	A1

```
INSERT INTO user
VALUES ('Burns', 'Saly',128,'A1')
```

Clustered index – příklad

kořen

klíč

blok

Blok 1001		
Bennet	Chet	1007
Karsen	Kit	1306
Smith	Ade	1062

vnitřní blok indexu

Blok 1007		
Bennet	Chet	1132
Fox	John	1133
Hunter	Leon	1127

Blok 1306		
Karsen	Kit	1198
Larn	Pard	1199
Peters	Mary	1200

listová úroveň indexu

Blok 1132			
Bennet	Chet	101	B2
Burns	Saly	128	A1
Claim	Dave	123	A1

Blok 1133			
Fox	John	100	A0
Greane	David	111	E3
Green	Mitch	125	B1
Greene	Joe	156	C3

Blok 1127			
Hunter	Leon	122	A3
Jetkins	Paul	124	A5

```
INSERT INTO user
VALUES ('Burns', 'Saly',128,'A1')
```

Použití indexů

- Select
  - Pokrývající query
- Group by
- Order by
- Join

- Nepoužívejte indexy pro malé tabulky, optimalizátor je stejně ignoruje
- Uvažujte o selektivitě indexů

Indexy – přístupové metody

- Typy indexů
  - B-tree
    - Clustered/nonclustered
  - Bitmapové indexy

- Standardní index
  - Hledání podle indexu
  - Scan nejnižší úrovně
  - Dotaz pokrytý indexem
  - Ignorování indexu – scan dat

- Clustered index
  - Hledání podle indexu
  - Scan dat od nalezeného řádku
  - Scan nejnižší úrovně (scan dat)

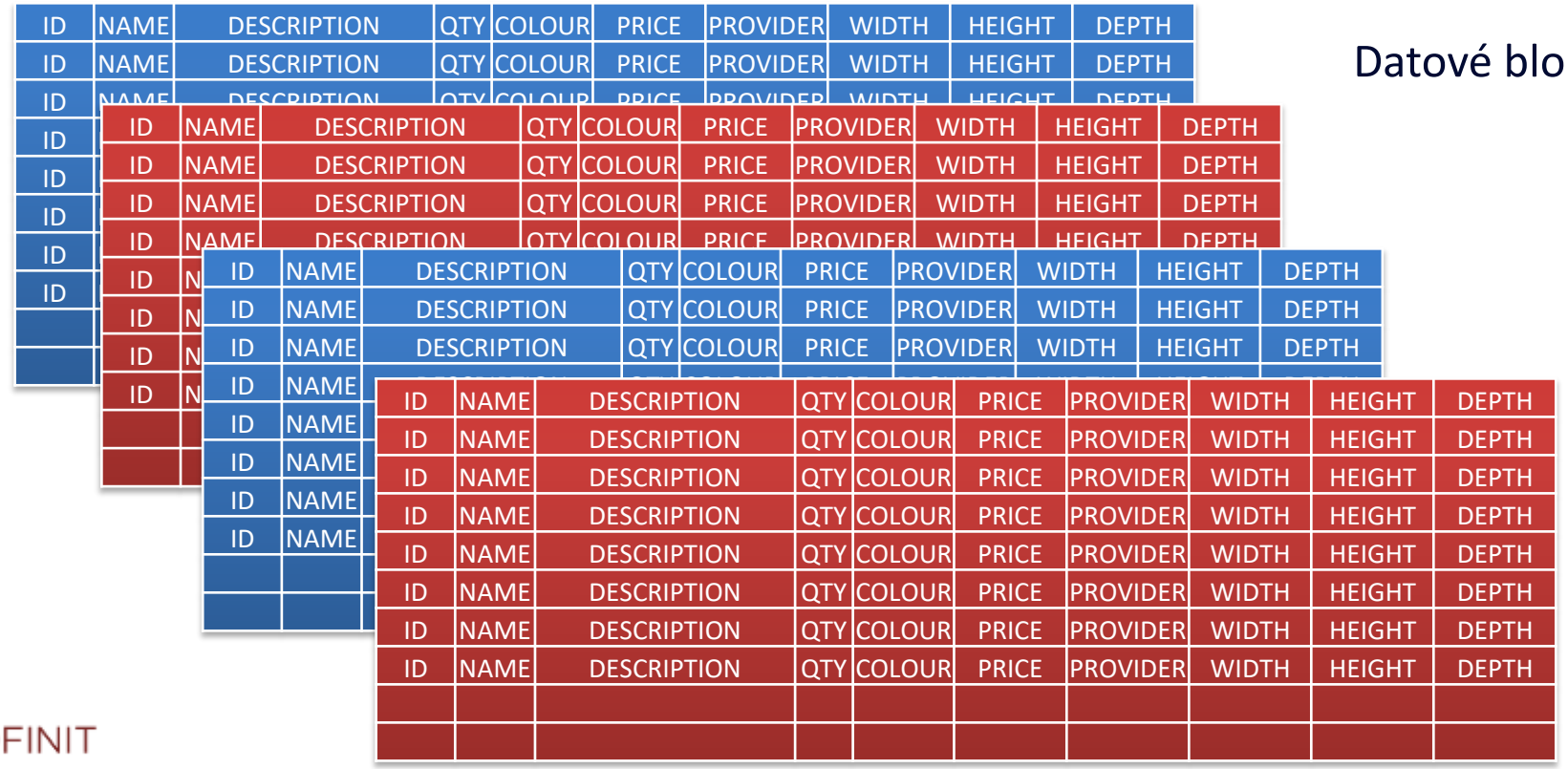
Data uložená po sloupcích

- Vertica
- SAP Sybase IQ
- Oracle – bitmapové indexy

Tabulka – relační uložení dat

ID	NUMBER
NAME	VARCHAR2(50 BYTE)
DESCRIPTION	VARCHAR2(500 BYTE)
QTY	NUMBER
COLOUR	VARCHAR2(20 BYTE)
PRICE	NUMBER
PROVIDER	VARCHAR2(40 BYTE)
WIDTH	NUMBER
HEIGHT	NUMBER
DEPTH	NUMBER

Struktura tabulky

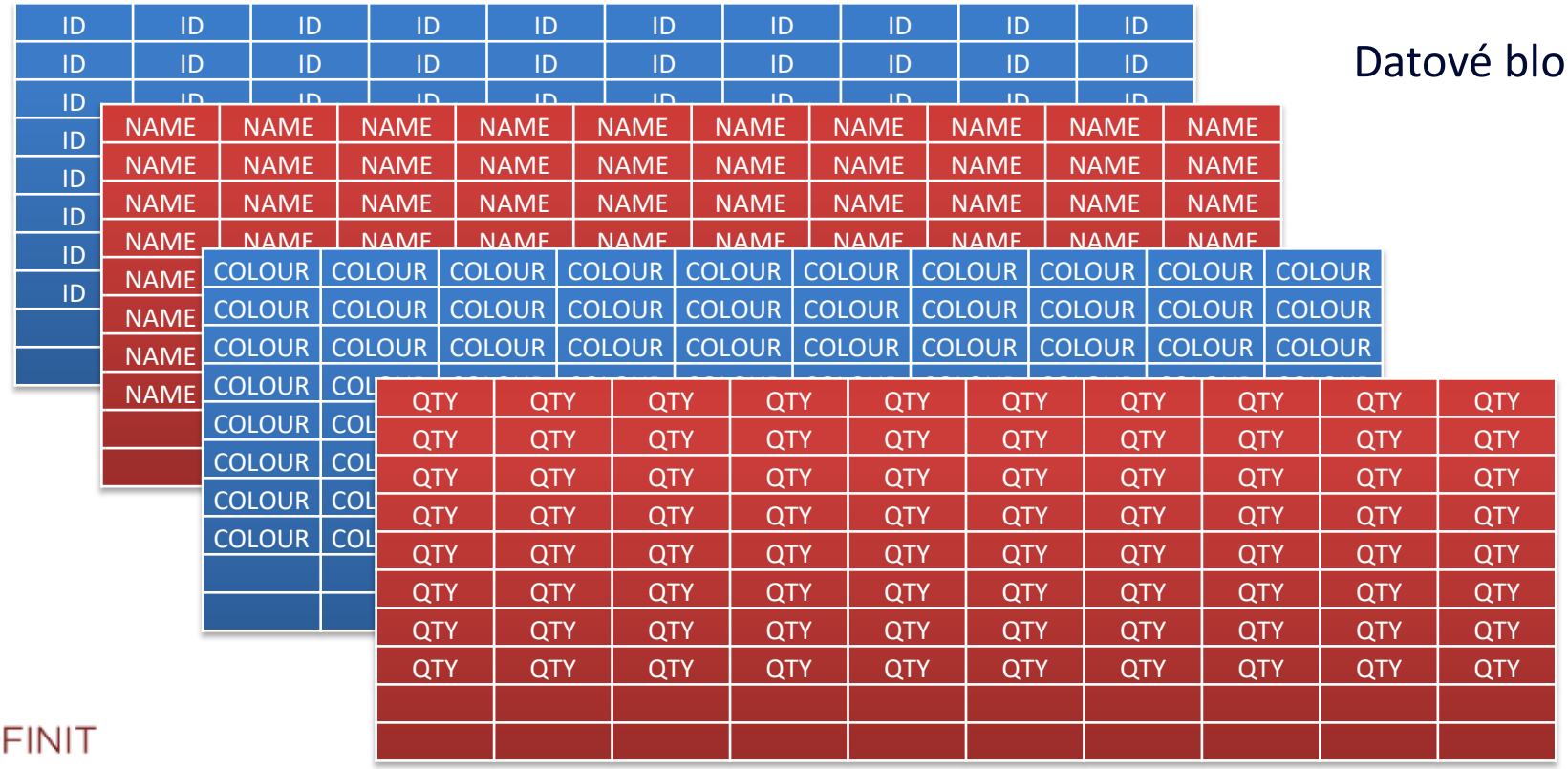


Datové bloky

Tabulka – data uložená po sloupcích

ID	NUMBER
NAME	VARCHAR2(50 BYTE)
DESCRIPTION	VARCHAR2(500 BYTE)
QTY	NUMBER
COLOUR	VARCHAR2(20 BYTE)
PRICE	NUMBER
PROVIDER	VARCHAR2(40 BYTE)
WIDTH	NUMBER
HEIGHT	NUMBER
DEPTH	NUMBER

Struktura tabulky



Datové bloky

Data uložená po sloupcích

Color
Red
Blue
Blue
Green
Green
Blue
Green
Red
Blue
Red

Green	Blue	Red
0	0	1
0	1	0
0	1	0
1	0	0
1	0	0
0	1	0
1	0	0
0	0	1
0	1	0
0	0	1

```
SELECT Count(*) FROM sale where color = 'Green'

SELECT Count(*) FROM sale where color in ('Green', 'Red')
```

Data uložená po sloupcích

Qty								64	32	16	8	4	2	1
37	0	1	0	0	1	0	1	0	1	0	0	1	0	1
102	1	1	0	0	1	1	0	1	1	0	0	1	1	0
21	0	0	1	0	1	0	1	0	0	1	0	1	0	1
123	1	1	1	1	0	1	1	1	1	1	1	0	1	1

Sloupec Binární zápis Index

```
SELECT SUM(qty) FROM sale
```

```
(2 * 64) + (3 * 32) + (2 * 16) + (1 * 8) +
  (3 * 4) + (2 * 2) + (3 * 1) = 283
```

Bitmapové indexy

- Rychlé pro sloupce s malou kardinalitou
- Rychlé pro operace na málo sloupcích

- Složitý update
  - Zamykání a rebuild velkých bloků
- Pomalé pro dotaz na jeden konkrétní řádek

Metody ukládání bitmapových indexů

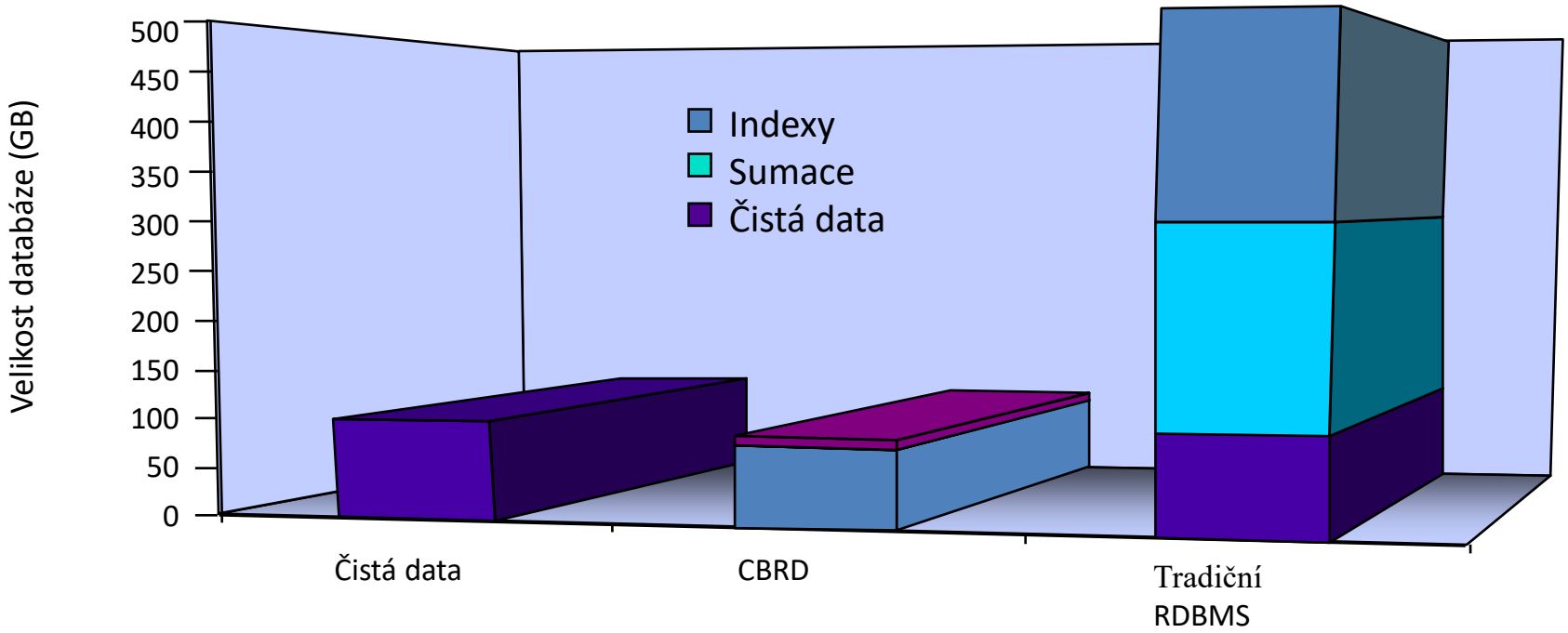
- Samé nuly nebo jedničky
  - Bloky se neukládají – pouze indikátor jejich existence
- Do 20% nul nebo jedniček
  - Data se kódují jako souvislé množiny hodnot
- Mezi 20% a 80% jedniček
  - Ukládá se skutečná mapa hodnot

Typy sloupcových indexů

- Mnoho různých typů
  - Více indexů na jednom sloupci

- Bitový index pro sloupce s malou kardinalitou
- Bitový index pro sloupce s velkou kardinalitou a malou selektivností
- Indexy pro sloupce s velkou kardinalitou G-Array (příbuzný B-tree)
- Prosté komprimované uložení dat (pro textové řetězce)
- Speciální indexy pro čas a datum
- Indexy pro joiny, porovnání a další operace

Sybase IQ – uložení dat a indexů



- Indexy jsou už data
- Nízké náklady na uložení dat
- Rychlé zpracování malého množství dat

Indexy

- Údržba indexů
  - Nejčastěji prováděná úprava na straně vývojářů i administrátorů
  - Přidání indexu zamyká tabulku – dopad na provoz
  - Mnoho indexů zpomaluji změny v datech
  - Indexy zabírají diskový prostor
  - Přegenerování stávajících indexů a přepočet statistik je součást standardní administrace serveru – vyžaduje čas
- Přidávání a rušení indexů
  - Je jednoduché zjistit, že se index v konkrétním příkazu používá
  - Nedá se zjistit, v kterých příkazech se index používá – definuje optimalizátor
  - Dá se zjistit, že se index někdy používá
  - Dá se zjistit, které indexy by byly vhodné pro konkrétní příkaz přidat
  - Nedá se zjistit, které indexy by šlo vyhodit
  - Nezapomenout na pokrývající indexy

Indexy – základní otázky

- Kdy začít řešit použití indexů a jejich přidávání?
  - Při logickém designu – základní indexy
  - Při fyzickém designu – na základě procesní matice
  - Při administraci – na základě sledování databáze a při optimalizaci
- Jaký je rozdíl mezi ASC a DESC?
  - Servery umí obecně lépe třídit pokud třídění odpovídá směru indexu.
- Je rozumné vytvářet indexy nad datumovým sloupcem?
  - Pokud se index neustále upravuje na jednom místě, na místě s aktuálním datem, může to vytvořit výkonnostní problém. Například při vkládání velkého množství aktuálních transakcí.
- Záleží na pořadí ve složených indexech?
  - Ano. Obecně se preferuje dávat sloupce s větší selektivitou na začátek. Pro pokrývající indexy zase na začátek sloupce z where klauzule.
- Platí: víc indexů = rychlejší databáze ?
  - Částečně. Indexy zrychlují dotazy, ale vyžadují režii při změnách.

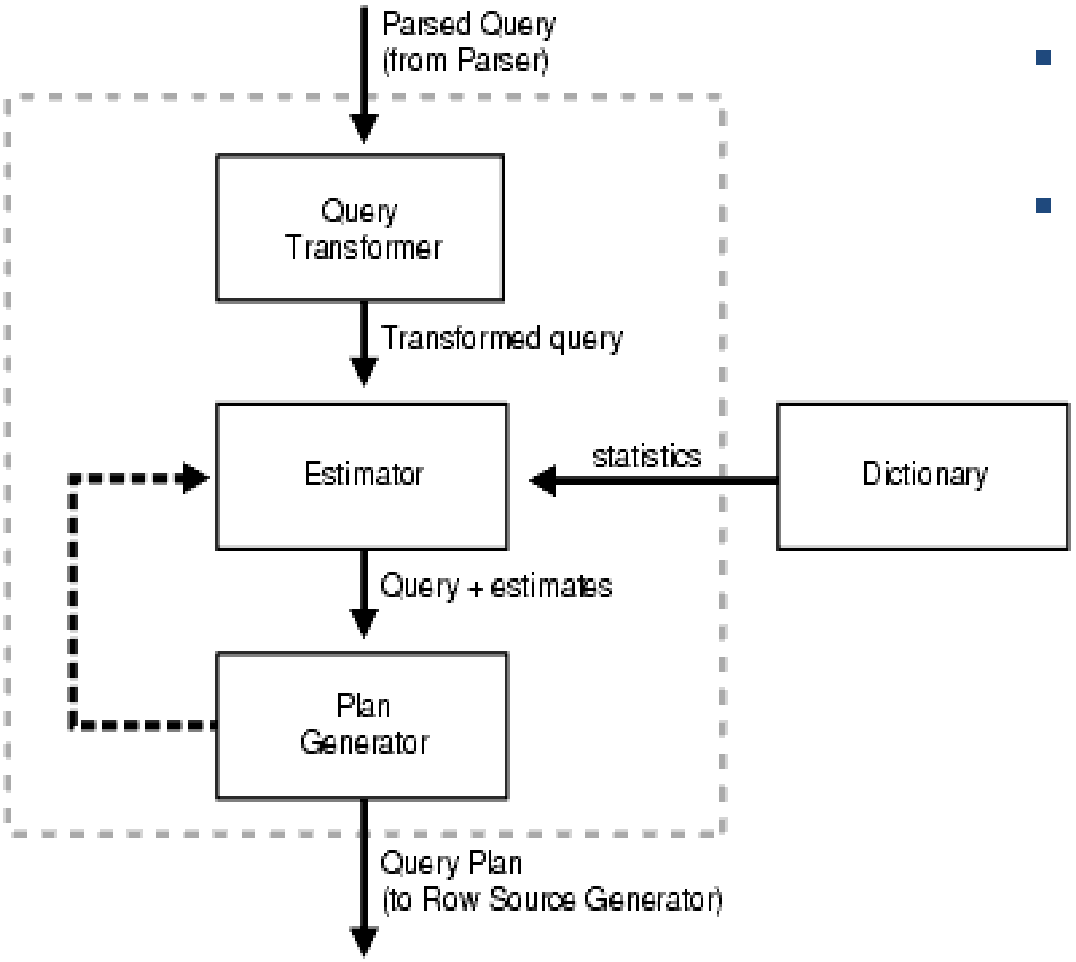
- 2

Optimalizátor

Optimalizace

- Pro SQL dotaz optimalizátor odhadne nejvhodnější exekuční plán
- Využití statistik – špatná statistika může i několikařádově zhoršit výkon, optimalizátor se rozhodne pro neefektivní plán
- Módy optimalizace
  - Nejkratší čas odpovědi
  - Největší průtok dat (default)
- Optimalizace používá statistiku
  - Metadata – struktura tabulek, indexů, …
  - Statistiky
- Optimalizaci lze ovlivnit používáním tipů (Hint)
- Optimalizace je obecně NP-úplný problém
- Optimalizace musí být podstatně kratší, než čas nutný k vykonání dotazu

Optimalizátor



- Rozdělení dotazu do kroků
- Volba metody přístupu k tabulkám
- Volba pořadí přístupu k tabulkám

Exekuční plán

- Zobrazení exekučního plánu
  - Tabulka PLAN_TABLE
  - Dynamický pohled V$SQL_PLAN
  - SQL analyzátor v Enterprise Manageru
  - EXPLAIN PLAN statement
  - Záložka Execution plan v SQL Developeru

```
SET AUTOTRACE ON
SELECT * from employee;

EXPLAIN PLAN FOR SELECT * FROM employee;
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Optimalizace

- Nejkratší čas odpovědi
  - Provedení dotazu tak, aby bylo prvních n řádků nalezeno nejrychleji
  - Vhodné pro interaktivní aplikace
- Největší průtok dat
  - Default mód pro Oracle
  - Provedení dotazu tak, aby byl celkový výsledek získán s minimem zdrojů
  - Vhodné pro aplikace v batch módu

Optimalizace

- Metodu lze zvolit inicializačním parametrem
  - Pro instance
    - Čas odpovědi
      - Init parameter: OPTIMIZER_MODE = first_rows_n
      - ALTER SYSTEM SET OPTIMIZER_MODE = first_rows_n
    - Průtok dat
      - Init parameter: OPTIMIZER_MODE = all_rows
      - ALTER SYSTEM SET OPTIMIZER_MODE = all_rows
  - Pro session
    - Čas odpovědi
      - ALTER SESSION SET OPTIMIZER_MODE = first_rows_n
    - Průtok dat
      - ALTER SESSION SET OPTIMIZER_MODE = all_rows
- n pro first_rows_n může být 1, 10, 100 nebo 1000

Statistiky

- Používá optimalizátor pro odhad nejlepšího plánu
- Příklady statistik
  - Tabulky (počet řádku, počet bloků, průměrná délka řádku)
  - Sloupce: počet různých hodnot, počet NULL, histogram
  - Indexy: Stránky, úrovně, cluster faktor
- Statistiky se neupravují ihned v rámci transakcí
- Přepočítávají se dávkově v rámci administrace
  - V pravidelných intervalech
  - Manuálně po změněn dat
- ORACLE - PL/SQL package DBMS_STATS
- MS SQL – příkaz update statistics

Hinty

- Hint se zapisuje inline příkazů
- Používat jen v odůvodněných případech
  - Se změnou dat se může stát, že dotaz začne být extrémně neefektivní
- Kategorie hintů
  - Cíl optimalizace
  - Metoda přístupu
  - Transformace dotazu
  - Pořadí joinů
  - Způsob joinů
  - Paralelní provedení
  - Další
- Execution plan
  - Přesný popis výpočtu
  - Možno přidělit k danému příkazu

Profiling

- Analýza chování běhu aplikací
  - Na úrovni celého datového serveru
  - Na úrovni jednotlivých příkazů
- Oracle - AWR report
  - Funguje pouze pro „named library units“
  - Informace se sbírají na úrovni PL/SQL VM
- Sybase – sp_sysmon procedura

- 2

Join


This is a WHERE clause
D
(DIM_DATE)
id
year = 1997
G
(DIM_GEOGRAPHY)
GROUP BY country
id
S
(DIM_STORE)
geography_id
id
F
(FACT_SALES)
date_id
product_id
store_id
SUM(units_sold)
P
(DIM_PRODUCT)
brand_id
id
product_category_id
C
(DIM_PRODUCT_CATEGORY)
id
product_category = 'tv'
This is one of the
GROUP BY columns
And this is the SUM
GROUP BY brand
id

JOIN

- Typy joinů
  - Nested loops
  - Hash JOIN
  - Merge JOIN
- Pořadí joinování
  - optimalizátor sám řeší pořadí joinů
  - n! možností pro n tabulek
- Bere v úvahu
  - velikost tabulek
  - počet kvalifikovaných sloupců (selektivitu where podmínek)
  - existenci indexů
  - existence statistik na join sloupcích (histogramy) – přesnější odhad
  - Strukturu tabulek (Index Organized Tables/Clustered index)
  - Partitioning tabulek (možnosti paralelizace výpočtu)

- 2

Bulk operace (Oracle)

Bulk operace

- náhrada FOR smyček s DML operacemi
- jeden DML příkaz pro FORALL (jeden „rozšířený“ příkaz)
- používá „bind array“ ze kterého se načítají hodnoty
  - libovolná kolekce (ale indexovaná celým číslem)
  - indexované sekvenčně (lze obejít přes INDICES OF / VALUES OF)
- výsledky (počet modifikovaných řádek)
  - tradiční „cursor attributes“
  - SQL%ROWCOUNT – celkový počet (nespolehlivé s LOG ERRORS)
  - SQL%BULK_ROWCOUNT – počet pro každý „příkaz“

Bulk operace - shrnutí

- asi nejzásadnější vlastnost týkající se výkonu
- něco za něco : vyšší složitost za vyšší výkon
- novější verze automatizují (10g „cursor for loops“)
- pozor na paměť - data se ukládají do PGA

Bulk operace

- zpracování velkého počtu řádek
- problém s přepínáním kontextu (SQL – PL/SQL)
- řešení – seskupit načítání a zápisy dat
- založeno na kolekcích
- FORALL
  - modifikace dat (INSERT, UPDATE, DELETE, MERGE)
  - zápis dat z kolekcí do databáze
- BULK COLLECT
  - načítání dat z tabulek do kolekcí
  - implicitní i explicitní dotazy

Příklad - tradiční

```
DECLARE
   TYPE data_t IS TABLE OF moje_tabulka%ROWTYPE;
   v_data data_t := data_t();
BEGIN

   FOR v_rec IN (SELECT * FROM moje_tabulka) LOOP
      v_data.EXTEND;
      v_data(v_data.LAST) := v_rec;
   END LOOP;

END;
```

Příklad - BULK

```
DECLARE
   TYPE data_t IS TABLE OF moje_tabulka%ROWTYPE;
   v_data data_t := data_t();
BEGIN

   SELECT * BULK COLLECT INTO v_data FROM moje_tabulka;

END;
```

Bulk collect

- pro dotazy i kurzory : INTO => BULK COLLECT INTO
- kolekce se vždy plní sekvenčně od hodnoty 1
- nevyhazuje výjimku NO_DATA_FOUND
- znáte max. počet záznamů => varray
  - LIMIT nebo velikost tabulky
  - pokud velikost nestačí, tak výjimka
- jinak nested table nebo index-by table
- optimální velikost LIMIT závisí na situaci
  - množství paměti, počet procesů, ...

Bulk collect

- nelze použít %NOTFOUND hned po fetchi
  - přesunout %NOTFOUND na konec smyčky
  - po fetchi spočítat elementy v kolekci (0 => konec)
  - na konci spočítat elementy v kolekci (< limit => konec)
- kdy převést tradiční fetch na BULK COLLECT
  - před 10g vždy (včetně „cursor for loops“)
  - od 10g není třeba převádět „cursor for loops“ pokud
    - neobsahují DML operace
    - běží dostatečně rychle

Příklad - tradiční

```
DECLARE
   TYPE data_t IS TABLE OF moje_tabulka%ROWTYPE;
   v_data data_t := data_t();
BEGIN
   v_data.EXTEND(10);
   v_data(1).id := 1; v_data(1).hodnota := 'a';
   /* atd pro další hodnoty */

   FOR idx IN v_data.FIRST .. v_data.LAST LOOP
      UPDATE moje_tabulka SET hodnota = v_data(idx).hodnota
                          WHERE    id = v_data(idx).id;
   END LOOP;
END;
```

Příklad - FORALL

```
DECLARE
   TYPE data_t IS TABLE OF moje_tabulka%ROWTYPE;
   v_data data_t := data_t();
BEGIN
   v_data.EXTEND(10);
   v_data(1).id := 1; v_data(1).hodnota := 'a';
   /* atd pro další hodnoty */

   FORALL idx IN v_data.FIRST .. v_data.LAST
      UPDATE moje_tabulka SET hodnota = v_data(idx).hodnota
                          WHERE    id = v_data(idx).id;

END;
```

- 8

ORM

ORM

- „Automatický“ převod mezi objektovým modelem v aplikaci a relačním modelem v databázi.
- Řeší jak strukturu tak manipulaci s daty
- Požadavek obecnosti řešení vynucuje použití obecných patternů pro struktury a algoritmy – nízká efektivita.
- ORM není schopno použít znalost kontextu.

- Často proto nutné obcházet z důvodů optimalizace.
- Programátor musí znát kontext a zároveň vědět jak ORM interně pracuje, jaký dopad bude mít změna parametrů.
- Kód píší vývojáři aplikace bez hlubší znalosti relačních databází.

- Object-relational impedance mismatch

Co si zapamatovat

- Kdy provádět optimalizaci
- Jaké jsou cíle optimalizace
- Jak pracuje optimalizátor a jaké informace využívá
- Co jsou nejčastější brzdy výkonu
- Jaký je rozdíl mezi strukturou tabulky s klastrovaným a neklastrovaným indexem
- K čemu slouží indexy a pro jaké přístupové metody k datům se používají
- Jak vypadají bitmapové indexy
- Co to je sloupcové uložení dat (sloupcové indexy)
- Kdy používat HINTs a kdy ne

Diskuse

- Otázky
- Poznámky
- Komentáře
- Připomínky