PA152: Efektivní využívání DB

9. Ladění dotazů

Vlastislav Dohnal

Poděkování

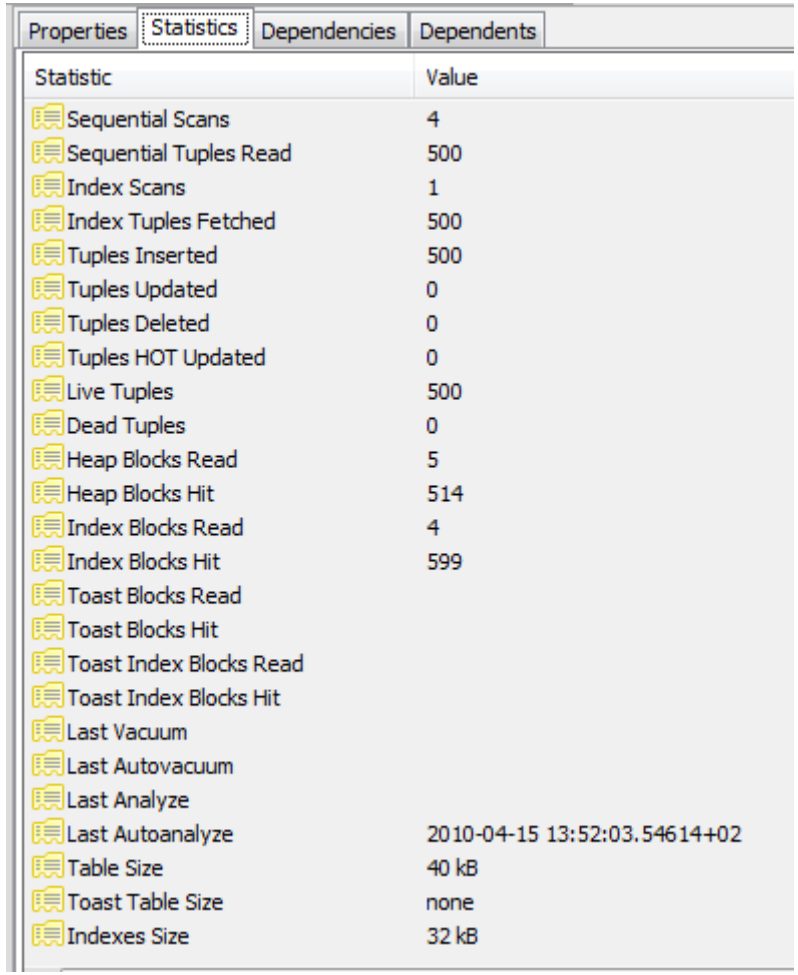
- Zdrojem materiálů tohoto předmětu jsou:
  - Přednášky CS245, CS345, CS345
    - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
    - Stanford University, California
  - Database Tuning (slides)
    - Dennis Shasha, Philippe Bonnet
    - Morgan Kaufmann, 1st edition, 440 pages, 2002
    - ISBN-13: 978-1558607538
    - http://www.databasetuning.org/

Příklad statistik PostgreSQL

- Připojte se k fakultní DB PostgreSQL
  - Návod viz první přednáška
- Ve schématu *xdohnal* jsou tabulky
  - *predmet*, *skupina, hotel*
    - Statistiky jak na relacích, tak i atributech.
  - Významy jednotlivých polí
    - http://www.postgresql.org/docs/8.1/interactive/view-pg-stats.html

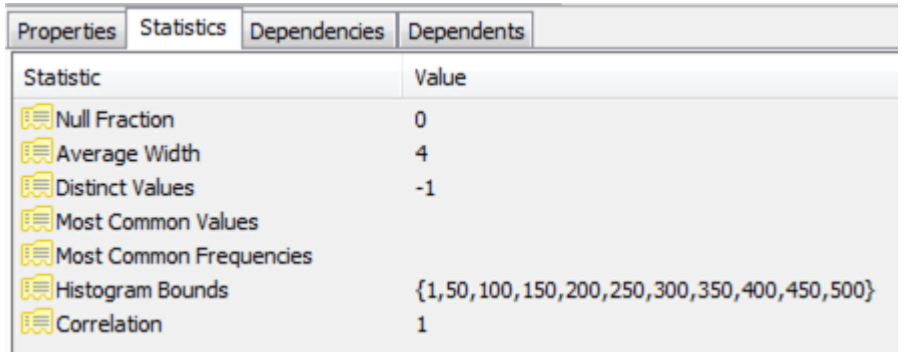
Příklad statistik PostgreSQL

- Tabulka hotel

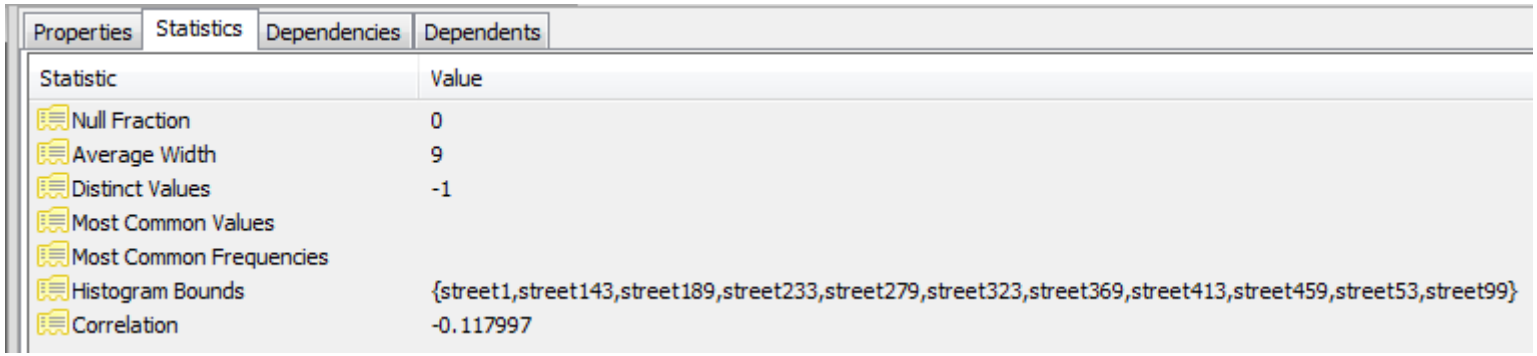


Příklad statistik PostgreSQL

- Atribut hotel.id

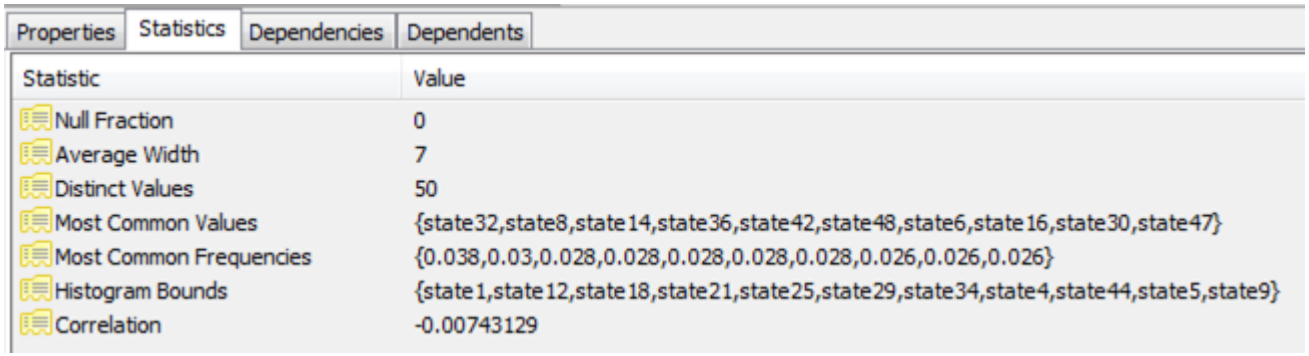


- Atribut hotel.name



Příklad statistik PostgreSQL

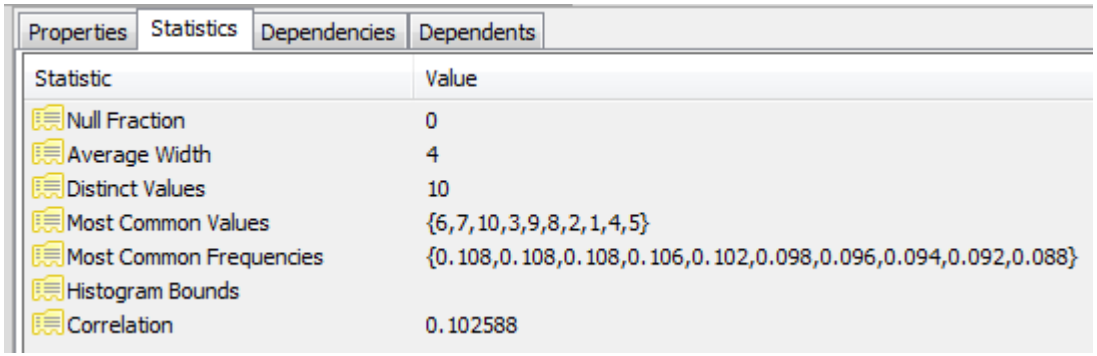
- Atribut hotel.state



Properties | Statistics | Dependencies | Dependents

Statistic	Value
Null Fraction	0
Average Width	7
Distinct Values	50
Most Common Values	{state32,state8,state14,state36,state42,state48,state6,state16,state30,state47}
Most Common Frequencies	{0.038,0.03,0.028,0.028,0.028,0.028,0.028,0.026,0.026,0.026}
Histogram Bounds	{state1,state12,state18,state21,state25,state29,state34,state4,state44,state5,state9}
Correlation	-0.00743129

- Atribut hotel.distance_to_center



Properties | Statistics | Dependencies | Dependents

Statistic	Value
Null Fraction	0
Average Width	4
Distinct Values	10
Most Common Values	{6,7,10,3,9,8,2,1,4,5}
Most Common Frequencies	{0.108,0.108,0.108,0.106,0.102,0.098,0.096,0.094,0.092,0.088}
Histogram Bounds	
Correlation	0.102588

Ladění dotazů

```
SELECT  s.RESTAURANT_NAME, t.TABLE_SEATING, to_char(t.DATE_TIME,'Dy, Mon FMDD') AS THEDATE,
to_char(t.DATE_TIME,'HH:MI PM') AS THETIME,to_char(t.DISCOUNT,'99') || '%' AS AMOUNTVALUE,t.TABLE_ID,
s.SUPPLIER_ID, t.DATE_TIME, to_number(to_char(t.DATE_TIME,'SSSSS')) AS SORTTIME
FROM TABLES_AVAILABLE t, SUPPLIER_INFO s,
      (SELECT s.SUPPLIER_ID, t.TABLE_SEATING, t.
       FROM TABLES_AVAILABLE t, SUPPLIER_INFO
       WHERE t.SUPPLIER_ID = s.SUPPLIER_ID
            and (TO_CHAR(t.DATE_TIME, 'MM/DD/YY
             or TO_NUMBER(TO_CHAR(sysdate, 'SSSS
            and t.NUM_OFFERS  > 0 and t.DATE_TIM
            and t.TABLE_SEATING = '2' and t.DATE_TIME between sysdate and (sysdate + 7)
            and to_number(to_char(t.DATE_TIME, 'SSSSS')) between 39600 and 82800
            and t.OFFER_TYPE = 'Discount'
       GROUP BY s.SUPPLIER_ID, t.TABLE_SEATING, t.DATE_TIME, t.OFFER_TYP) u
WHERE t.SUPPLIER_ID=s.SUPPLIER_ID and u.SUPPLIER_ID=s.SUPPLIER_ID and t.SUPPLIER_ID=u.SUPPLIER_ID
      and t.TABLE_SEATING = u.TABLE_SEATING and t.DATE_TIME = u.DATE_TIME
      and t.DISCOUNT = u.AMOUNT and t.OFFER_TYPE = u.OFFER_TYPE
      and (TO_CHAR(t.DATE_TIME, 'MM/DD/YYYY') != TO_CHAR(sysdate, 'MM/DD/YYYY')
            or TO_NUMBER(TO_CHAR(sysdate, 'SSSSS'))  < s.NOTIFICATION_TIME - s.TZ_OFFSET)
      and t.NUM_OFFERS > 2 and t.DATE_TIME > SYSDATE and s.CITY = 'SF'
      and t.TABLE_SEATING = '2'  and t.DATE_TIME between sysdate and (sysdate + 7)
      and to_number(to_char(t.DATE_TIME, 'SSSSS')) between 39600 and 82800 and t.OFFER_TYPE = 'Discount'
ORDER BY AMOUNTVALUE DESC,  t.TABLE_SEATING ASC, upper(s.RESTAURANT_NAME) ASC,
            SORTTIME ASC, t.DATE_TIME ASC
```

Provedení je příliš pomalé ...

1) Jak je dotaz vyhodnocován?
2) Jak jej lze urychlit?

Plán dotazu

Výstup příkazu EXPLAIN v Oracle

```
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=165 Card=1 Bytes=106)
   1    0   SORT (ORDER BY) (Cost=165 Card=1 Bytes=106)
   2    1     NESTED LOOPS (Cost=164 Card=1 Bytes=106)
   3    2       NESTED LOOPS (Cost=155 Card=1 Bytes=83)
   4    3         TABLE ACCESS (FULL) OF 'TABLES_AVAILABLE' (Cost=72 Card=1 Bytes=28)
   5    3         VIEW
   6    5           SORT (GROUP BY) (Cost=83 Card=1 Bytes=34)
   7    6             NESTED LOOPS (Cost=81 Card=1 Bytes=34)
   8    7               TABLE ACCESS (FULL) OF 'TABLES_AVAILABLE' (Cost=72 Card=1 Bytes=24)
   9    7               TABLE ACCESS (FULL) OF 'SUPPLIER_INFO' (Cost=9  Card=20 Bytes=200)
  10    2       TABLE ACCESS (FULL) OF 'SUPPLIER_INFO' (Cost=9 Card=20 Bytes=460)
```

Operátor

Přístupová metoda

Cena provedení

Monitorování dotazů

- Možnosti, jak objevit pomalý dotaz:
  - Vyžaduje příliš mnoho přístupů na disk
    - Např. dotaz na přesnou shodu používá table-scan.
  - Nevhodný plán dotazu
    - Vhodné (existující) indexy nejsou použity
  - Databáze umožňují logování „dlouhých“ dotazů

Ladění dotazu

- První přístup ke zrychlení jsou lokální změny
- Globální změna
  - Vytvoření indexu
  - Změna schéma
  - Rozdělení transakcí
  - …
- Lokální změna
  - Přepsání dotazu ovlivní pouze daný dotaz

Přepisování dotazů

- Příklad:
  - Employee(ssnum, name, manager, dept, salary, numfriends)
    - Shlukovaný index na *ssnum*
      - Tj. určuje uspořádání souboru
    - Neshlukované indexy: (i) *name* a (ii) *dept*
  - Student(ssnum, name, degree_sought, year)
    - Shlukovaný index na *ssnum*
    - Neshlukovaný index na *name*
  - Tech(dept, manager, location)
    - Shlukovaný index na *dept*

Přepisování dotazů

- Techniky
  - Použití indexů
  - Rušení nadbytečných DISTINCT
  - (Korelované) poddotazy
  - Dočasné tabulky
  - Podmínky spojení
  - Používání HAVING
  - Používání pohledů (VIEW)
  - Uložené pohledy (materialized views)

Používání indexů

- Optimalizace dotazů nemusí použít index pokud jsou používány:
  - Aritmetické výrazy

    WHERE salary/12 >= 4000;
  - Podřetězce

    SELECT * FROM employee
    WHERE SUBSTR(name, 1, 1) = ‘G’;
  - Porovnávání atributů různých datových typů
  - Víceatributové indexy
  - Porovnání na NULL

Rušení nadbytečných DISTINCT

- Dotaz:
  - Najdi zaměstnance pracující v oddělení *informační systémy*. Ve výsledku nechme duplicity.
  - SELECT DISTINCT ssnum
    FROM employee
    WHERE dept = ‘information systems’
- DISTINCT není nutný
  - *ssnum* je primární klíč v *employee*

Rušení nadbytečných DISTINCT

- Dotaz:
  - Vypiš čísla *ssnum* všech zaměstnanců z technického oddělení. Ve výsledku nechme opakování.
  - SELECT DISTINCT ssnum
    FROM employee, tech
    WHERE employee.dept = tech.dept
- Je DISTINCT nutný?

Rušení nadbytečných DISTINCT

- Dotaz:
  - SELECT DISTINCT ssnum
    FROM employee, tech
    WHERE employee.dept = tech.dept
- Je DISTINCT nutný?
  - *ssnum* je primární klíč v *employee*
  - *dept* je primární klíč v *tech*
  - → každý zaměstnanec se spojí s nejvýše jedním záznamem z relace *tech.*
  - → DISTINCT není potřeba

Rušení nadbytečných DISTINCT

- Vztah mezi DISTINCT, primárními klíči a spojeními lze popsat:
  - Relace *T* je *privilegovaná,* pokud atributy vracené příkazem SELECT obsahují primární klíč.
  - Nechť *R* není privilegovaná relace.
  - Když *R* je spojena s relací *S* podle rovnosti primárního klíče *R* a nějakého atribut(ů) z *S*, pak *R je závislá na S*.
  - Relace „*záviset na*“ je tranzitivní:
    - R_1 závisí na R_2 a R_2 závisí na R_3, pak R_1 závisí na R_3.

Rušení nadbytečných DISTINCT

- Ve výsledku příkazu SELECT nebudou duplicity (bez DISTINCT), pokud platí alespoň jedno z:
  - Každá relace ve FROM je privilegovaná.
  - Každá neprivilegovaná relace závisí na nějaké privilegované.

Nadbytečný DISTINCT (1)

- Dotaz:
  - SELECT ssnum
    FROM employee, tech
    WHERE employee.manager = tech.manager
- *Employee* je privilegovaná
- Je *tech* privilegovaná?
  - Ne.
- Závisí *tech* na *employee*?
  - Ne, protože atribut *manager* není primárním klíčem *tech*.

Nadbytečný DISTINCT (2)

- Dotaz:
  - SELECT ssnum, tech.dept
    FROM employee, tech
    WHERE employee.manager = tech.manager
- *Employee* je privilegovaná
- Je *tech* privilegovaná?
  - Ano.
- Výsledky se neopakují

Nadbytečný DISTINCT (3)

- Dotaz:
  - SELECT student.ssnum
    FROM student, employee, tech
    WHERE student.name = employee.name
    AND employee.dept = tech.dept;
- *Student* je privilegovaná
- *Employee* není privilegovaná a nezávisí na žádné z ostatních relací.
- → DISTINCT je nutný.

Typy vnořených dotazů

- Nekorelované dotazy s agregační funkcí uvnitř
  - SELECT ssnum FROM employee WHERE salary > (SELECT avg(salary) FROM employee)
- Nekorelované dotazy bez agregační funkce
  - SELECT ssnum FROM employee WHERE dept in (SELECT dept FROM tech)

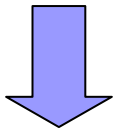
Typy vnořených dotazů

- Korelované s agregační funkcí
  - SELECT ssnum FROM employee e1
    WHERE salary =
      (SELECT avg(e2.salary)
       FROM employee e2, tech
       WHERE e2.dept = e1.dept
             AND e2.dept = tech.dept)
- Korelované bez agregační funkce
  - Neobvyklé (resp. lze napsat pomocí spojení)

Přepsání nekorelovaných dotazů bez agregace

1. Relace z obou FROM dej dohromady
2. IN nahraď rovností (=)
3. Vybírané atributy se nemění

```
SELECT ssnum FROM employee
WHERE dept in (select dept from tech)
```



```
SELECT ssnum
FROM employee, tech
WHERE employee.dept = tech.dept
```

Přepsání nekorelovaný dotazů bez agregace

- Problém s duplicitami:
  - SELECT avg(salary) FROM employee WHERE manager in (select manager from tech)
  - SELECT avg(salary) FROM employee, tech WHERE employee.manager = tech.manager
- Druhý dotaz může vracet zaměstnance vícekrát
  - Pokud stejný manažer vede více oddělení.
- Řešením je pomocná tabulka
  - Kde pomocí DISTINCT eliminujeme duplicity.

Přepsání korelovaných dotazů

- Dotaz:
  - Najdi zaměstnance technických oddělení, kteří vydělávají průměrnou mzdu svého oddělení.

```
SELECT ssnum
FROM employee e1
     WHERE salary = (SELECT avg(e2.salary
                    FROM employee e2, tech
                    WHERE e2.dept = e1.dept
                      AND e2.dept = tech.dept);
```

Přepsání korelovaných dotazů

```
INSERT INTO temp
        SELECT avg(salary) as avsalary, employee.dept
        FROM employee, tech
        WHERE employee.dept = tech.dept
        GROUP BY employee.dept;

SELECT ssnum
FROM employee, temp
WHERE salary = avsalary
    AND employee.dept = temp.dept
```

Přepsání korelovaných dotazů

- Dotaz:
  - Najdi zaměstnance technických oddělení, kteří mají stejně kamarádů jako kolegů ve svém oddělení.

```
SELECT ssnum
FROM employee e1
WHERE numfriends = COUNT(
                    SELECT e2.ssnum
                    FROM employee e2, tech
                    WHERE e2.dept = tech.dept
                                AND e2.dept = e1.dept);
```

Přepsání korelovaných dotazů

```
INSERT INTO temp
        SELECT COUNT(ssnum) as numcolleagues,
                employee.dept
        FROM employee, tech
        WHERE employee.dept = tech.dept
        GROUP BY employee.dept;

SELECT ssnum
        FROM employee, temp
        WHERE numfriends = numcolleagues
                AND employee.dept = temp.dept;
```

Vznikl zde problém v COUNT?

Problém v COUNT?

- Příklad:
  - Helena nepracuje v technickém oddělení.
  - V původním dotazu by se její přátelé porovnávali s COUNT(Ø)=0. V případě, že Helena nemá přátele, zůstane ve výběru.
  - V přepsaném dotazu by se záznam Heleny ve výsledku neobjevil.
    - Pomocná tabulka bude obsahovat pouze počty pro technická oddělení.
- Toto je omezení při přepisování korelovaných dotazů s COUNT.

Používání pomocných tabulek

- Dotaz:
  - Pro zaměstnance oddělení informačních systémů, kteří mají plat > 40000, vypiš jejich číslo *ssnum* a umístění.
  - INSERT INTO temp
    SELECT *
    FROM employee
    WHERE salary >= 40000
  - SELECT ssnum, location
    FROM temp
    WHERE temp.dept = ‘information systems’
- Toto řešení nebude optimální
  - Nelze využít index na *dept*
  - Optimalizátor dotazů takový index na *temp* nemá.

Používání HAVING

- Důvod zavedení
  - Zkrácení dotazů, které filtrují podle výsledku agregačních funkcí
  - Ve WHERE nelze použít agregační funkci
  - V klauzuli HAVING ano
- Příklad
  - SELECT avg(salary), dept
    FROM employee
    GROUP BY dept
    HAVING avg(salary) > 10 000;

Používání HAVING

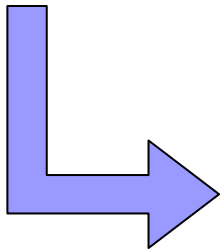
- Jiný příklad

```
SELECT avg(salary), dept
FROM employee
GROUP BY dept
HAVING count(ssnum) > 100;
```

Používání HAVING

- Nepoužívat HAVING
  - Pokud lze zapsat ve WHERE.

```
SELECT avg(salary) as avgsalary, dept
FROM employee
GROUP BY dept
HAVING dept = 'information systems';
```



```
SELECT avg(salary) as avgsalary, dept
FROM employee
WHERE dept= 'information systems'
GROUP BY dept;
```

Používání pohledů

```
CREATE VIEW techlocation AS
    SELECT ssnum, tech.dept, location
    FROM employee, tech
    WHERE employee.dept = tech.dept;

SELECT location
FROM techlocation
WHERE ssnum = 43253265;
```

- Optimalizátor dotazů provede nahrazení pohledu jeho definicí

Používání pohledů

- Výsledkem dostaneme:

```
SELECT location
FROM employee, tech
WHERE employee.dept = tech.dept
      AND ssnum = 43253265;
```

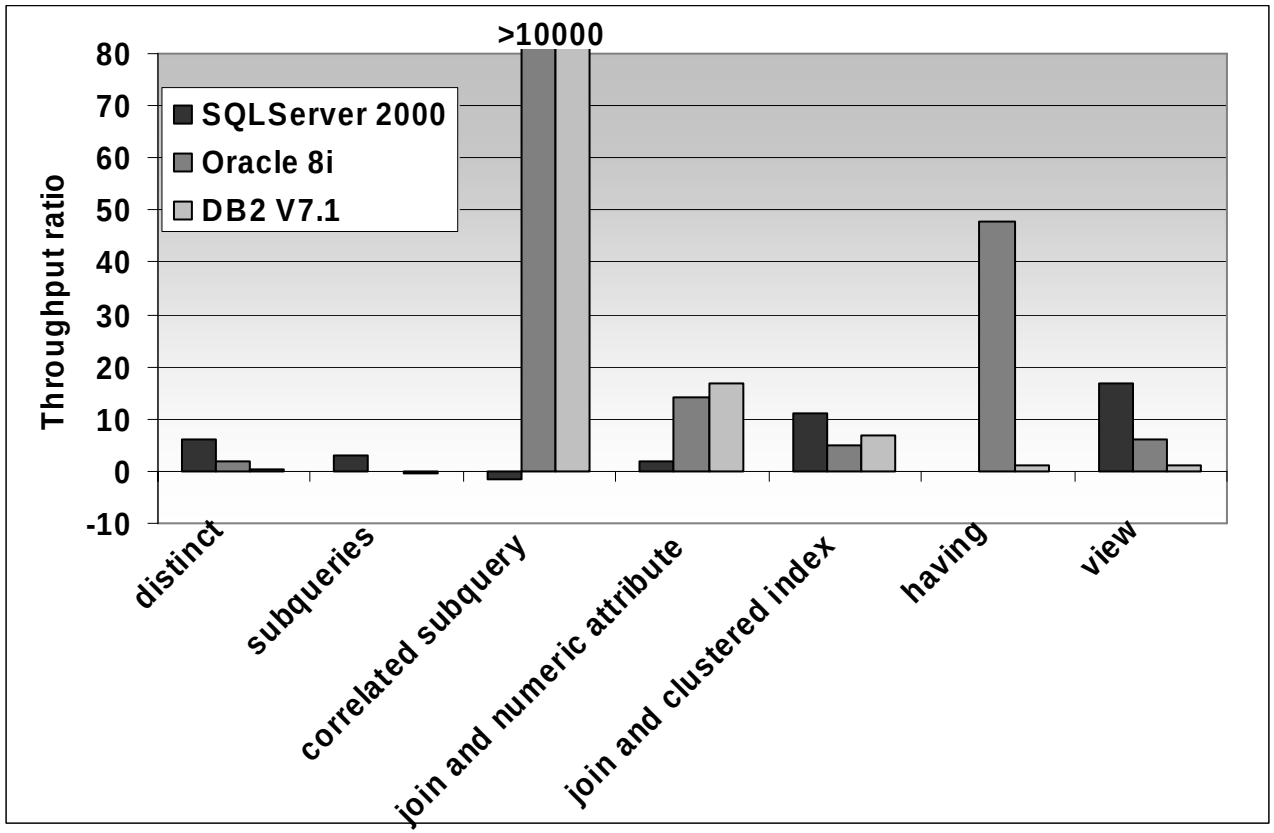
Používání pohledů

- Příklad v PostgreSQL:
  - CREATE VIEW hotels_in_city AS
    SELECT city, COUNT(*) AS count
    FROM hotel
    GROUP BY city;
- Použití pohledu
  - SELECT * FROM hotels_in_city
    WHERE count > 8;

Používání pohledů

- Příkaz EXPLAIN
  - EXPLAIN SELECT * FROM hotels_in_city;
  - EXPLAIN SELECT * FROM hotels_in_city WHERE count > 8;
    - Porovnejte s
      EXPLAIN SELECT city, COUNT(*)
      FROM hotel GROUP BY city
      HAVING COUNT(*) > 8;

Přepisování dotazů: výkonnostní vliv



100k zaměstnanců, 100k studentů, 10 tech. oddělení

Optimalizace agregačních funkcí

- Příklad:
  - Evidence objednávek obchodního řetězce
    - Order(ordernum, itemnum, quantity, purchaser, vendor)
    - Item(itemnum, description, price)
    - Shlukované indexy nad *itemnum* pro *Order* a *Item*
  - Každých 5 minut se provádí dotazy:
    - Celková cena objednaného zboží jistého výrobce (vendor).
    - Celková cena objednaného zboží nějakým obchodem (purchaser).

Optimalizace agregačních funkcí

- Dotazy:
  - SELECT vendor, sum(quantity*price)
    FROM order, item
    WHERE order.itemnum = item.itemnum
    GROUP BY vendor;
  - SELECT purchaser, sum(quantity*price)
    FROM order, item
    WHERE order.itemnum = item.itemnum
    GROUP BY purchaser;
  - Cena dotazů?
    - → jsou drahé

Optimalizace agregačních funkcí

- Jak zrychlit?
  - Definice pohledů?
    - → nepomůže
  - Ukládat výsledky do pomocných tabulek?
    - → pomůže

Optimalizace agregačních funkcí

- Vytvoříme tabulky
  - OrdersByVendor(vendor, amount)
  - OrdersByPurchaser(purchaser, amount)
- Tabulky se musí aktualizovat
  - Kdy aktualizovat?
    - Po každé změně *order*, popř. *item*?
      - Realizovat pomocí triggerů (spouští)
    - Periodicky po určitém čase znovu vytvořit
  - Náklady na aktualizaci
    - Musí být menší než náklady na původní dotazy.

Uložené (materializované) pohledy

- Výsledek pohledu je uložený v tabulce
  - Automatická aktualizace databází
    - Obvykle…
  - Použití i v dotazech, které daný pohled nepoužívají
    - Optimalizátor dotazů přepisuje dotaz

Uložené (materializované) pohledy

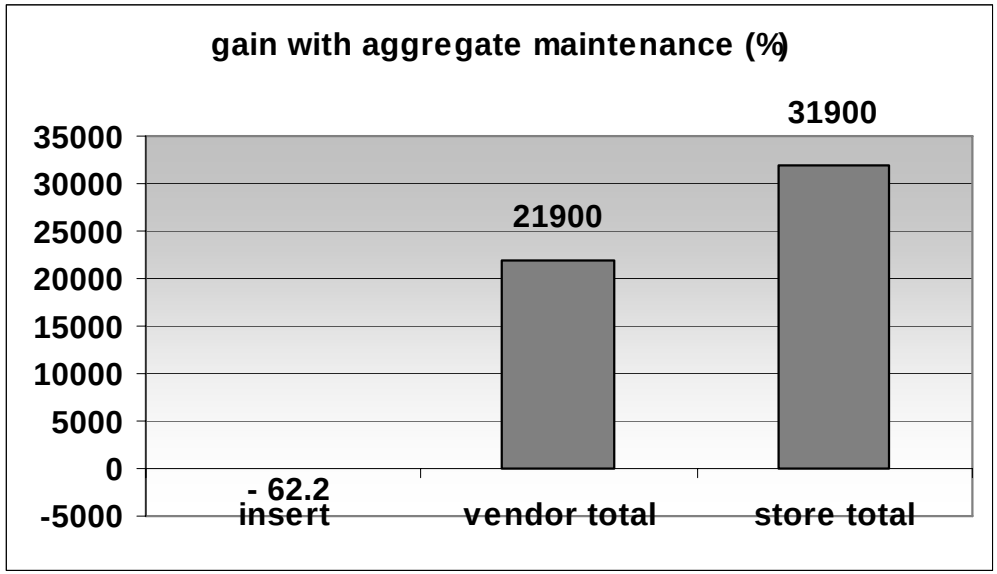
- Např. Oracle
  - CREATE MATERIALIZED VIEW OrdersByVendor
    BUILD IMMEDIATE REFRESH COMPLETE
    ENABLE QUERY REWRITE
    AS
    SELECT vendor, sum(quantity*price) AS amount
    FROM order, item
    WHERE order.itemnum = item.itemnum
    GROUP BY vendor;

Uložené (materializované) pohledy

- Příklad
  - QUERY REWRITE
  - Dotaz:
    - SELECT vendor, sum(quantity*price) AS amount
      FROM order, item
      WHERE order.itemnum = item.itemnum
      AND vendor='Apple'
      GROUP BY vendor;
    - Použije se pohled OrdersByVendor
      - SELECT vendor, amount FROM OrdersByVendor
        WHERE vendor='Apple';

Uložené (materializované) pohledy

- Příklad
  - SQLServer, implementováno pomocí triggerů
  - 1m objednávek – 5 obchodů, 20 výrobců
  - 10k položek



Databázové spouště (triggers)

- Spoušť je uložené procedura
  - Kód používající SQL příkazy prováděný při výskytu nějaké události.
- Události:
  - DML – insert, update, delete
  - Časové (nebývá časté)
  - DDL – definice tabulek, …

Databázové spouště (triggers)

- Nezávislé na aplikaci
  - Protože jsou prováděny samotným DB serverem.
- Bez spouští musí být vše řešeno aplikací
- Přinášejí dodatečné náklady
  - Mohou vkládat do dalších tabulek, …
  - Spouštění omezovat podmínkami
    - Např. při aktualizaci ceny, počtu objednaných položek
    - Ne při aktualizaci popisu položky, …

Globální změny

- Vytvoření indexu
- Změna schéma
  - Viz další přednáška
- Rozdělení relací
  - Viz další přednáška
- …

Používání indexů

- Malá tabulka
  - Indexy jsou vytvořeny
  - Přesto nejsou používány
- Příklad
  - predmet(kod, nazev, kredity)
  - SELECT COUNT(*) FROM predmet;
    - Výsledek 3
  - SELECT * FROM predmet WHERE kod='MA102';
    - Použije se table-scan.

Vytváření indexů

- Sekvenční čtení tabulky
  - Všechny záznamy jsou kontrolovány
  - → pomalé
- Vytvoření indexu
  - Zrychlí SELECT
  - Zpomalí INSERT, UPDATE, DELETE
    - Index se musí aktualizovat

Vliv indexu na náklady

- Neplatí:
  - Čím více indexů, tím rychlejší zpracování!
    - Teoreticky platné pouze pro SELECT.
- Každý index zpomaluje aktualizace
  - Nutné aktualizovat kromě relace i index
  - Pozor:
    - INSERT INTO tabulka SELECT …
    - DELETE FROM tabulka WHERE …

Příklad odhadu nákladů

- Relace
  - StarsIn(movieTitle, movieYear, starName)
- Q_{movies}
  - SELECT movieTitle, movieYear FROM StarsIn WHERE starName='jméno';
- Q_{stars}
  - SELECT starName FROM StarsIn WHERE movieTitle='název' AND movieYear='rok';
- Insert
  - INSERT INTO StarsIn VALUES ('název', 'rok', 'jméno');

Příklad odhadu nákladů

- Předpoklady:
  - B(StarsIn) = 10 bloků
  - Každá hvězda hraje průměrně ve 3 filmech.
  - Každý film má průměrně 3 hvězdy.
  - Relace není nijak uspořádaná.
    - Pokud je index, pak 3 čtení z disku.
  - Aktualizace indexu – 1 čtení a 1 zápis bloku
  - Vkládání do relace – 1 čtení a 1 zápis bloku
    - I bez indexu nehledáme volný blok.

Příklad odhadu nákladů

- Počty čtení a zápisů pro jednotlivé situace
  - Pravděpodobnost provádění operací
    - $Q_{movies}=p_1$, $Q_{stars}=p_2$, Insert=1 - p_1 - p_2
  - Situace1: $p_1 = p_2 = 0.1 \rightarrow$ bez indexů
  - Situace2: $p_1 = p_2 = 0.4 \rightarrow$ oba indexy

Akce	Žádný index	Index *starName*	Index *movieTitle, movieYear*	Oba indexy
Q_{movies}	10	4	10	4
Q_{stars}	10	10	4	4
Insert	2	4	4	6
Prům. náklady	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

Optimalizace indexů

1. Stanovit dávku příkazů
   - Tj. způsob vytížení
   - Analýzou logů zjistit typy dotazů, aktualizací a jejich četnosti
2. Navrhnout různé indexy
   - Optimalizátor nechat odhadnou cenu vyhodnocení dávky příkazů
   - Vybrat konfiguraci s nejmenší cenou
3. Vytvořit indexy, které minimalizují cenu

Optimalizace indexů

- Ad bod 2
  - Mám sadu možných indexů
  - Začni bez indexů
  - Opakuj
    - Pro každý navrhovaný index, vypočítej cenu
    - Vytvoř index s nejvyšším vylepšením ceny
      - A používej jej v další iteraci
    - Opakuj, dokud byl nějaký index vytvořený.

- Celý proces lze dělat i automaticky
  - MS AutoAdmin (http://research.microsoft.com/en-us/projects/autoadmin/default.aspx)
  - MS Index Tuning Wizard (S. Chaudhuri, V. Narasayya: *An efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server*. Proceedings of VLDB Conference, 1997)
  - Oracle 10g (http://www.oracle-base.com/articles/10g/AutomaticSQLTuning10g.php)

Referenční integrita

- Vytvoření cizího klíče neznamená index na atributech
- Příklad v PostgreSQL (db.fi.muni.cz)
  - Hotel – primární klíč *id*
  - Room – primární klíč *id*, cizí klíč *hotel_id*
    - V(Room, hotel_id) = 6
- Dotazy (zajímá nás výsledek EXPLAIN)

```
SELECT * FROM hotel WHERE id=2;
SELECT * FROM room WHERE hotel_id=2 AND number=1;
```

Referenční integrita

- Dotaz

```
SELECT * FROM room WHERE hotel_id=2 AND number=1;
```

- Bez indexu

```
Seq Scan on room  (cost=0.00..6741.89 rows=103 width=22)
  Filter: ((hotel_id = 2) AND (number = 1))
```

- Vytvoříme index nad *hotel_id*

```
CREATE INDEX hotel_id_fkey ON room (hotel_id);

Bitmap Heap Scan on room  (cost=981.00..3784.38 rows=103 width=22)
  Recheck Cond: (hotel_id = 2)
  Filter: (number = 1)
  ->  Bitmap Index Scan on hotel_id_fket  (cost=0.00..980.97 rows=52892 width=0)
        Index Cond: (hotel_id = 2)
```

Referenční integrita

- Cizí klíče mohou velmi zpomalit i mazání
- Příklad:
  - DELETE FROM hotel WHERE id=500;
  - Cizí klíč v *room* odkazuje na tabulku *hotel*
  - Při mazání se musí v tabulce *room* kontrolovat přítomnost záznamů *hotel_id=500*
- Doporučení
  - Vytvářet na cizích klíčích indexy