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/

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.DA
       FROM TABLES_AVAILABLE t, SUPPLIER_INFO s
       WHERE t.SUPPLIER_ID = s.SUPPLIER_ID
            and (TO_CHAR(t.DATE_TIME, 'MM/DD/YYYY'
             or TO_NUMBER(TO_CHAR(sysdate, 'SSSSS')
            and t.NUM_OFFERS  > 0 and t.DATE_TIME >
            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

- Co je pomalý dotaz?
  - Vyžaduje příliš mnoho přístupů na disk
    - vysoké *costs* v plánu (explain)
    - Např. dotaz na přesnou shodu používá table-scan.
  - Nevhodný plán dotazu
    - Vhodné (existující) indexy nejsou použity
- Jak je objevit?
  - Databáze umožňují logování „dlouhých“ dotazů
  - …

Ladění dotazu

- Lokální změna = přepsání dotazu
  - První přístup ke zrychlení dotazu
  - Ovlivní pouze daný dotaz
- Globální změna
  - Vytvoření indexu
  - Změna schématu
  - Rozdělení transakcí
  - …

Přepisování dotazů

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

Přepisování dotazů

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

Používání indexů

- Optimalizátor dotazů nemusí použít index, pokud jsou používány:
  - Aritmetické výrazy

    WHERE salary/12 >= 4000;

    WHERE inserted + 1 = current_date;
  - Funkce

    SELECT * FROM employee
      WHERE SUBSTR(name, 1, 1) = ‘G’;

    … WHERE to_char(inserted, 'YYYYMM') = '201704'
  - Porovnávání atributů různých datových typů
  - Víceatributové indexy
  - Porovnání na NULL

Používání indexů (pokr.)

- Agregační funkce MAX(*A*), MIN(*A*)
  - resp. ORDER BY *A* LIMIT 1
  - použití funkcí v A
  - Např.
    - conn_log (log_key, sim_imsi, time, car_key, pda_imei, gsmnet_id, method, program_ver)

Plus a secondary index on (sim_imsi,time)

A. SELECT **max(time AT TIME ZONE 'UTC')** AS time
FROM conn_log
WHERE sim_imsi=‘23001234567890123’ AND
time> '2016-02-28 10:50:00.122 UTC' AND
method='U' AND program_ver IS NOT NULL;

B. SELECT **time AT TIME ZONE 'UTC‘**
FROM (SELECT **max(time)** AS time
FROM conn_log
WHERE sim_imsi=‘23001234567890123’ AND
time>'2016-02-28 10:50:00.122 UTC' AND
method='U' AND program_ver IS NOT NULL) AS x;

C. SELECT **max(time) AT TIME ZONE 'UTC'** AS time …
(cont. from query A.)

Používání indexů (pokr.)

```
QUERY PLAN  (QUERY A.)
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=19412.69..19412.70 rows=1 width=8) (actual time=36.415..36.415 rows=1 loops=1)
  ->  Append  (cost=0.00..19385.45 rows=5448 width=8) (actual time=36.410..36.410 rows=0 loops=1)
        ->  Seq Scan on conn_log  (cost=0.00..0.00 rows=1 width=8) (actual time=0.003..0.003 rows=0 loops=1)
              Filter: ((program_ver IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone) AND (sim_imsi = '23001234567890123'::bpchar) AND
                                                                                                                                   (method = 'U'::bpchar))
        ->  Index Scan using conn_log_imsi_time_y2016m02 on conn_log_y2016m02  (cost=0.56..8.58 rows=1 width=8) (actual time=28.464..28.464 rows=0 loops=1)
              Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
              Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
        ->  Bitmap Heap Scan on conn_log_y2016m03  (cost=194.11..14125.36 rows=3969 width=8) (actual time=2.586..2.586 rows=0 loops=1)
              Recheck Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
              Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
              ->  Bitmap Index Scan on conn_log_imsi_time_y2016m03  (cost=0.00..193.12 rows=4056 width=0) (actual time=2.584..2.584 rows=0 loops=1)
                    Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
        ->  Bitmap Heap Scan on conn_log_y2016m04  (cost=71.87..5243.35 rows=1476 width=8) (actual time=5.346..5.346 rows=0 loops=1)
              Recheck Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
              Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
              ->  Bitmap Index Scan on conn_log_imsi_time_y2016m04  (cost=0.00..71.50 rows=1507 width=0) (actual time=5.342..5.342 rows=0 loops=1)
                    Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
        ->  Index Scan using conn_log_imsi_time_y2016m05 on conn_log_y2016m05  (cost=0.14..8.16 rows=1 width=8) (actual time=0.009..0.009 rows=0 loops=1)
              Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
              Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
Planning time: 4.159 ms
Execution time: 36.535 ms
```

Používání indexů (pokr.)

```
QUERY PLAN  (QUERY B.)
------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Subquery Scan on x  (cost=5.98..6.01 rows=1 width=8) (actual time=0.162..0.163 rows=1 loops=1)
  ->  Result  (cost=5.98..5.99 rows=1 width=0) (actual time=0.159..0.160 rows=1 loops=1)
        InitPlan 1 (returns $0)
          ->  Limit  (cost=1.87..5.98 rows=1 width=8) (actual time=0.158..0.158 rows=0 loops=1)
                ->  Merge Append  (cost=1.87..22424.61 rows=5449 width=8) (actual time=0.156..0.156 rows=0 loops=1)
                      Sort Key: conn_log."time"
                      ->  Index Scan Backward using conn_log_imsi_time on conn_log  (cost=0.12..8.15 rows=1 width=8) (actual time=0.004..0.004 rows=0 loops=1)
                            Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                            Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
                      ->  Index Scan Backward using conn_log_imsi_time_y2016m02 on conn_log_y2016m02  (cost=0.56..8.58 rows=1 width=8)
                                                                                                        (actual time=0.069..0.069 rows=0 loops=1)
                            Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                            Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
                      ->  Index Scan Backward using conn_log_imsi_time_y2016m03 on conn_log_y2016m03  (cost=0.56..16225.91 rows=3969 width=8)
                                                                                                        (actual time=0.046..0.046 rows=0 loops=1)
                            Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                            Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
                      ->  Index Scan Backward using conn_log_imsi_time_y2016m04 on conn_log_y2016m04  (cost=0.43..6033.60 rows=1477 width=8)
                                                                                                        (actual time=0.035..0.035 rows=0 loops=1)
                            Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                            Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
                      ->  Index Scan Backward using conn_log_imsi_time_y2016m05 on conn_log_y2016m05  (cost=0.14..8.17 rows=1 width=8)
                                                                                                        (actual time=0.002..0.002 rows=0 loops=1)
                            Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                            Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
Planning time: 3.137 ms
Execution time: 0.317 ms
```

Používání indexů (pokr.)

```
QUERY PLAN  (QUERY C.)
------------------------------------------------------------------------------------------------------------------------------------------------------------
Result  (cost=5.98..5.99 rows=1 width=0) (actual time=0.186..0.186 rows=1 loops=1)
  InitPlan 1 (returns $0)
    ->  Limit  (cost=1.87..5.98 rows=1 width=8) (actual time=0.182..0.182 rows=0 loops=1)
          ->  Merge Append  (cost=1.87..22424.63 rows=5450 width=8) (actual time=0.181..0.181 rows=0 loops=1)
                Sort Key: conn_log."time"
                ->  Index Scan Backward using conn_log_imsi_time on conn_log  (cost=0.12..8.15 rows=1 width=8) (actual time=0.005..0.005 rows=0 loops=1)
                      Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                      Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
                ->  Index Scan Backward using conn_log_imsi_time_y2016m02 on conn_log_y2016m02  (cost=0.56..8.58 rows=1 width=8)
                                                                                         (actual time=0.070..0.070 rows=0 loops=1)
                      Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                      Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
                ->  Index Scan Backward using conn_log_imsi_time_y2016m03 on conn_log_y2016m03  (cost=0.56..16225.91 rows=3969 width=8)
                                                                                         (actual time=0.064..0.064 rows=0 loops=1)
                      Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                      Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
                ->  Index Scan Backward using conn_log_imsi_time_y2016m04 on conn_log_y2016m04  (cost=0.43..6033.60 rows=1478 width=8)
                                                                                         (actual time=0.037..0.037 rows=0 loops=1)
                      Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                      Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
                ->  Index Scan Backward using conn_log_imsi_time_y2016m05 on conn_log_y2016m05  (cost=0.14..8.17 rows=1 width=8)
                                                                                         (actual time=0.003..0.003 rows=0 loops=1)
                      Index Cond: ((sim_imsi = '23001234567890123'::bpchar) AND ("time" IS NOT NULL) AND ("time" > '2016-02-28 11:50:00.122+01'::timestamp with time zone))
                      Filter: ((program_ver IS NOT NULL) AND (method = 'U'::bpchar))
Planning time: 3.094 ms
Execution time: 0.309 ms
```

Rušení nadbytečných DISTINCT

- Dotaz:
  - Najdi zaměstnance pracující v oddělení *informační systémy*. Ve výsledku nechceme 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 nějakého technického oddělení. Ve výsledku nechceme opakování.
  - SELECT DISTINCT ssnum
    FROM employee, tech
    WHERE employee.dept = tech.dept
- Je DISTINCT nutný?

Employee(ssnum, name, manager, dept, salary, numfriends)
Tech(dept, manager, location)

Rušení nadbytečných DISTINCT

- Dotaz:

  Employee(<u>ssnum</u>, name, manager, dept, salary, numfriends)
  Tech(<u>dept</u>, manager, location)

  - 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:
  - Definice „*privilegovanost*“
    - Relace *T* je *privilegovaná,* pokud atributy vrácené příkazem SELECT obsahují její primární klíč.
  - Definice relace „*záviset na*“
    - Nechť *R* <u>není privilegovaná</u> relace.
    - Když *R* je spojena s relací *S* podle rovnosti <u>primárního klíče *R*</u> a odpovídajících atributu(ů) z *S*, pak <u>*R je závislá na S*</u>. (např. S má cizí klíč do R)
  - 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

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

Nadbytečný DISTINCT (1)

- Dotaz:

  Employee(<u>ssnum</u>, name, manager, dept, salary, numfriends)
  Tech(<u>dept</u>, manager, location)

  - SELECT DISTINCT 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)

Employee(ssnum, name, manager, dept, salary, numfriends)
Tech(dept, manager, location)

- Dotaz:
  - SELECT DISTINCT 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 DISTINCT 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.

Employee(ssnum, name, manager, dept, salary, numfriends)
Student(ssnum, name, degree_sought, year)
Tech(dept, manager, location)

Vnořené dotazy

- Příkaz SELECT obsahující další SELECT jako svoji část
  - SELECT employee_number, name
    FROM employees AS X
    WHERE salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department = X.department);
  - SELECT employee_number, name,
    (*SELECT AVG(salary) FROM employees*
    WHERE department = X.department) AS department_average
    FROM employees AS X;

Přepisování vnořených dotazů

- Důvod:
  - Optimalizátor dotazů nemusí vždy správně fungovat na některých vnořených dotazech
  - Typicky:
    - Nekorelované dotazy bez agregační funkce
    - Korelované dotazy

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)
    - Tzv. semi-join

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)

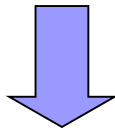
Typy vnořených dotazů

- Korelované bez agregační funkce
  - Neobvyklé (resp. lze napsat pomocí spojení)
    - Semi-join dotazy mohou být vyhodnoceny neefektivně
  - Příklad semi-join dotazů (2 různé):
    - SELECT ssnum FROM employee
      WHERE dept IN
      (SELECT dept FROM tech
      WHERE tech.manager=employee.manager)
    - SELECT ssnum FROM employee
      WHERE EXISTS (SELECT 1 FROM tech WHERE
      employee.manager = tech.manager)

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 nekorelovaných 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 alespoň průměrnou mzdu svého oddělení.

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

Přepsání korelovaných dotazů

```
CREATE TEMPORARY TABLE temp ( … ) ON COMMIT DROP;

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

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

Přepsání korelovaných dotazů

```
SELECT ssnum
FROM employee as E,
        (SELECT avg(salary) as avsalary, tech.dept
         FROM tech, employee
         WHERE tech.dept = employee.dept
         GROUP BY tech.dept) as AVG
WHERE salary >= avsalary AND E.dept = AVG.dept
```

Přepsání korelovaných dotazů

- Dotaz:
  - Najdi zaměstnance technických oddělení, kteří kamarádí s celým oddělením
    (mají stejně kamarádů jako pracovníků ve svém oddělení).

```
SELECT ssnum
FROM employee e1
WHERE numfriends = (
            SELECT COUNT(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 numworkers,
        employee.dept
    FROM tech, employee
    WHERE tech.dept = employee.dept
    GROUP BY tech.dept;

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

Vznikl zde problém v COUNT?

Problém v COUNT?

- Příklad:
  - Helena pracuje v ekonomickém oddělení.
  - V původním dotazu by se její přátelé porovnávali s COUNT(∅)=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.

Přepsání korelovaných dotazů

- Anti-joins
  - SELECT * FROM Tech WHERE dept NOT IN (SELECT dept FROM employee)
    - Problém s případnou NULL v employee.dept
  - SELECT * FROM Tech WHERE NOT EXISTS (SELECT 1 FROM employee.dept=tech.dept)
- Problémy
  - Nepoužívání join algoritmu
  - Používání index join na příliš velkém množství záznamů

Přepisování dotazů

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

Používání pomocných tabulek

- Dotaz:
  - Pro zaměstnance oddělení informačních systémů, kteří mají plat > 40000, vypiš jejich *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* v *employee*
  - 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

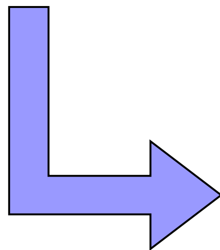
- Jiný příklad

```
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;
```

(řádek „GROUP BY dept;“ je na snímku přeškrtnutý)

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;
  - SELECT * FROM hotels_in_city
    WHERE city='city174'

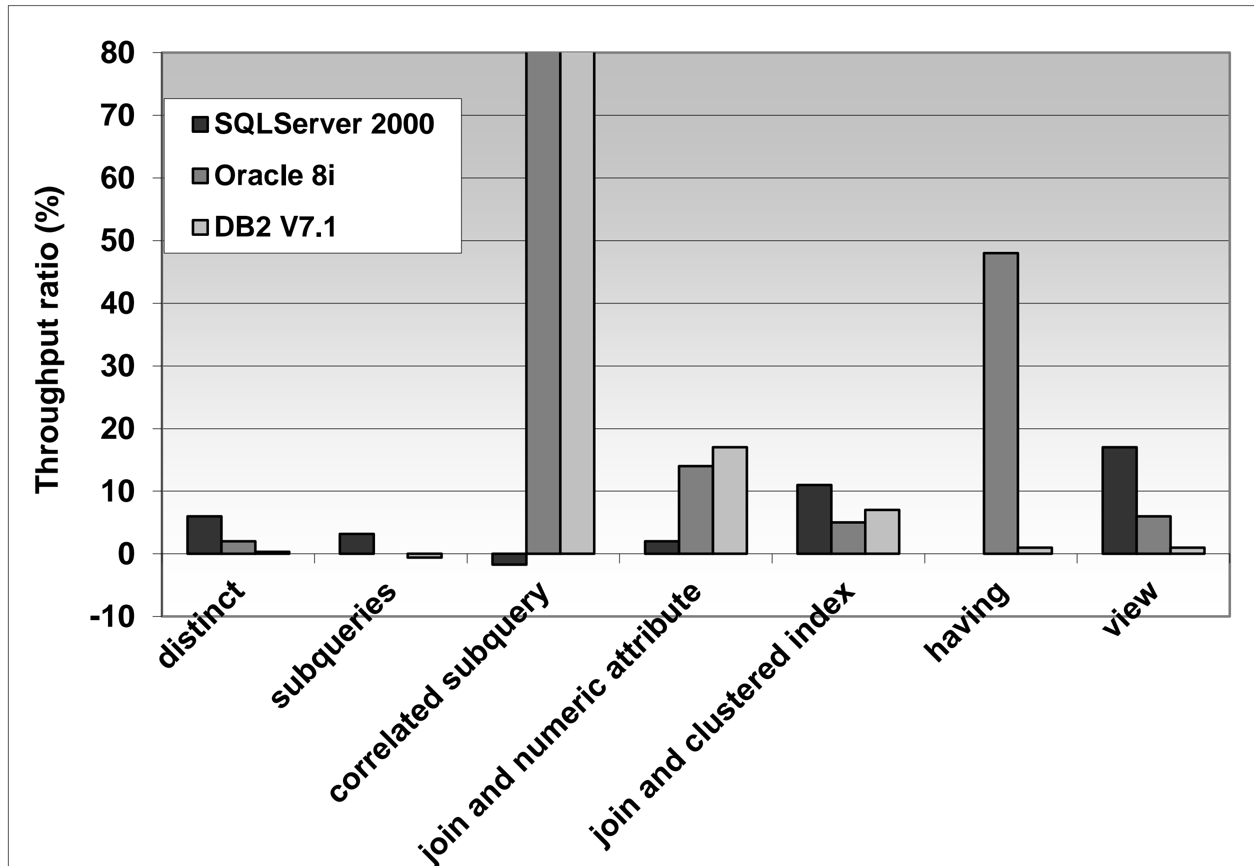
Používání pohledů

- Příkaz EXPLAIN
  - EXPLAIN SELECT * FROM hotels_in_city;
  - EXPLAIN SELECT * FROM hotels_in_city WHERE city='city174';
    - Porovnejte s:

```
EXPLAIN SELECT * FROM
   (SELECT lower(city) as city, COUNT(*) AS cnt
    FROM hotel GROUP BY city HAVING COUNT(*) > 3) x
WHERE city='city174';
```

Přepisování dotazů: výkonnostní vliv

>10 000



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

Optimalizace agregačních funkcí

- Příklad:
  - Evidence objednávek obchodního řetězce
    - Order(<u>ordernum</u>, itemnum, quantity, purchaser, vendor)
    - Item(<u>itemnum</u>, 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?
  - $\rightarrow$ jsou drahé

Optimalizace agregačních funkcí

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

Optimalizace agregačních funkcí

- Vytvoříme tabulky
  - OrdersByVendor(<u>vendor</u>, amount)
  - OrdersByPurchaser(<u>purchaser</u>, 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

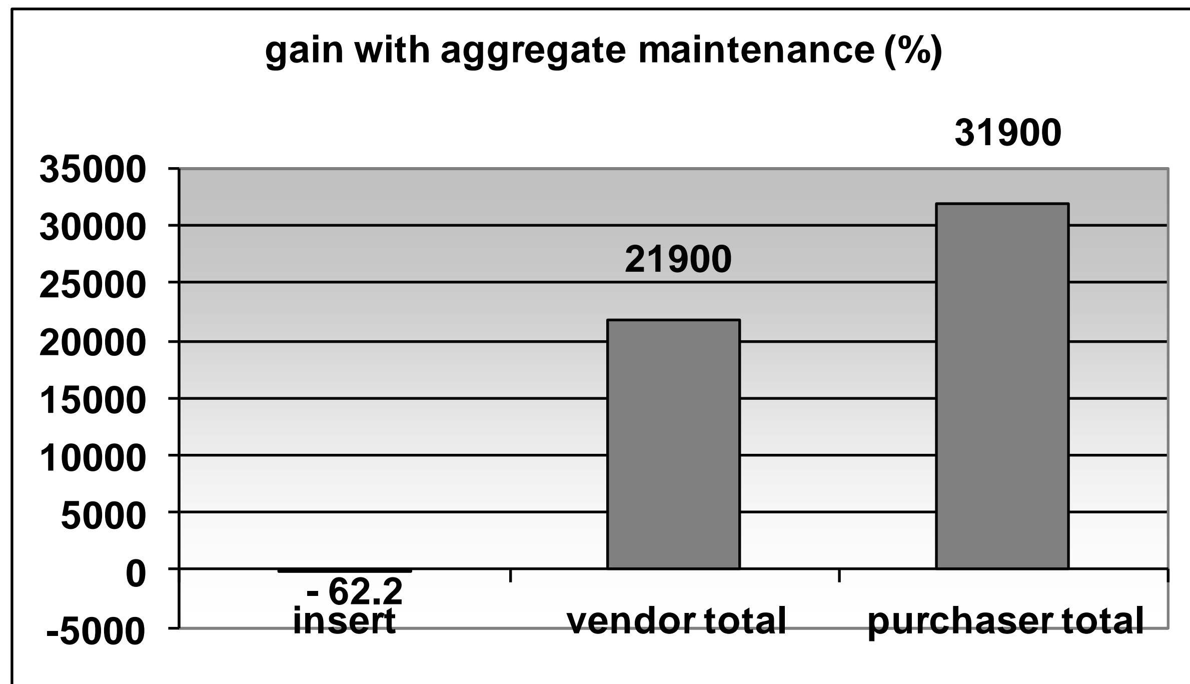
- Např. Oracle
  - CREATE MATERIALIZED VIEW OrdersByVendor
    BUILD IMMEDIATE REFRESH COMPLETE
    ENABLE QUERY REWRITE
    AS
    SELECT vendor, sum(quantity*price) AS amount
    FROM order, item
    WHERE order.itemnum = item.itemnum
    GROUP BY vendor;

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';
    - Použije se pohled OrdersByVendor
      - SELECT vendor, amount FROM OrdersByVendor WHERE vendor='Apple';

Uložené (materializované) pohledy

- Příklad
  - SQLServer, implementováno pomocí triggerů
  - 1m orders – 5 purchasers and 20 vendors
  - 10k items



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(<u>kod</u>, nazev, kredity)
  - SELECT COUNT(*) FROM predmet;
    - Výsledek: 5
  - SELECT * FROM predmet
    WHERE kod='MA102';
    - Použije se table-scan (seq scan)

Vytváření indexů

- Sekvenční čtení tabulky (table scan / seq scan)
  - Všechny záznamy jsou kontrolovány
  - $\rightarrow$ pomalé
- Vytvoření indexu (index scan)
  - 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 …

Vliv indexu na náklady: příklad

- Relace
  - StarsIn(movieTitle, movieYear, starName)
- Q_{movies}
  - SELECT movieTitle, movieYear FROM StarsIn
    WHERE starName=‘name’;
- Q_{stars}
  - SELECT starName FROM StarsIn
    WHERE movieTitle=‘title’ AND movieYear=year;
- Insert
  - INSERT INTO StarsIn VALUES (‘title’, year, ‘name’);

Vliv indexu na náklady: příklad

- Předpoklady:
  - B(StarsIn) = 10 bloků
  - Každý herec hraje průměrně ve 3 filmech.
  - Každý film má průměrně 3 hvězdy.
  - Relace není nijak uspořádaná.
    - Pokud je index, pak 1 čtení z disku pro každý záznam.

  - Prohledání indexu
    - 1 čtení
  - Aktualizace indexu
    - 1 čtení a 1 zápis bloku
  - Vkládání do relace
    - 1 čtení a 1 zápis bloku
      - Tj. nehledáme volný blok (jak s indexem, tak bez)

Vliv indexu na náklady: příklad

- Počty čtení a zápisů pro jednotlivé situace
  - Pravděpodobnost provádění operací
    - $Q_{movies} = p_1$, $Q_{stars} = p_2$, $Insert = p_i = 1 - p_1 - p_2$

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	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

- Situace1: $p_1 = p_2 = 0.1 \rightarrow$ bez indexů
- Situace2: $p_1 = p_2 = 0.4 \rightarrow$ oba indexy

Optimalizace indexů

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

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
      - používej jej v dalších iteracích
    - Opakuj, dokud byl nějaký index vytvořen.

- 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) & the best 10-year paper in 2007!
  - Oracle 10g (http://www.oracle-base.com/articles/10g/AutomaticSQLTuning10g.php)
  - pgAnalyze, HypoPg

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 (výstup z EXPLAIN SELECT…)

```
Seq Scan on room  (cost=0.00..8750.89 rows=105 width=22)
  Filter: ((hotel_id = 2) AND (number = 1))
```

- Vytvoříme index nad *hotel_id*

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

Bitmap Heap Scan on room  (cost=974.87..5782.99 rows=105 width=22)
  Recheck Cond: (hotel_id = 2)
  Filter: (number = 1)
  ->  Bitmap Index Scan on room_hotel_id_fkey  (cost=0.00..974.84 rows=52608 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

Kombinace indexů

- Dotaz SELECT * FROM room WHERE hotel_id=2 AND number=1;
- Pouze index nad *hotel_id*

```
"Bitmap Heap Scan on room  (cost=960.80..5756.77 rows=103 width=22)"
"  Recheck Cond: (hotel_id = 2)"
"  Filter: (number = 1)"
"  ->  Bitmap Index Scan on room_hotel_id_fkey  (cost=0.00..960.77 rows=51798 width=0)"
"        Index Cond: (hotel_id = 2)"
```

- Pouze index nad *number*

```
"Bitmap Heap Scan on room  (cost=13.02..1688.30 rows=103 width=22)"
"  Recheck Cond: (number = 1)"
"  Filter: (hotel_id = 2)"
"  ->  Bitmap Index Scan on room_number_idx  (cost=0.00..12.99 rows=628 width=0)"
"        Index Cond: (number = 1)"
```

Kombinace indexů

- Dotaz SELECT * FROM room WHERE hotel_id=2 AND number=1;
- Index nad *hotel_id, number*

```
"Bitmap Heap Scan on room  (cost=5.34..366.14 rows=103 width=22)"
"  Recheck Cond: ((hotel_id = 2) AND (number = 1))"
"  ->  Bitmap Index Scan on room_hotel_id_number_fkey  (cost=0.00..5.31 rows=103 width=0)"
"        Index Cond: ((hotel_id = 2) AND (number = 1))"
```

- Dva indexy nad *hotel_id* a *number*

```
"Bitmap Heap Scan on room  (cost=974.07..1334.86 rows=103 width=22)"
"  Recheck Cond: ((number = 1) AND (hotel_id = 2))"
"  ->  BitmapAnd  (cost=974.07..974.07 rows=103 width=0)"
"        ->  Bitmap Index Scan on room_number_idx  (cost=0.00..12.99 rows=628 width=0)"
"              Index Cond: (number = 1)"
"        ->  Bitmap Index Scan on room_hotel_id_fkey  (cost=0.00..960.77 rows=51798 width=0)"
"              Index Cond: (hotel_id = 2)"
```

Index s reverzním klíčem

- Specialita Oracle
- Zvýšení průchodnosti indexu
  - počet vkládání / aktualizací za čas
- Idea:
  - Hodnoty klíče v indexu používat reverzně
  - $\rightarrow$ hodnoty ze sekvencí jsou rozptýleny
    - Např. 12345 a 12346 $\rightarrow$ 54321 a 64321
  - $\rightarrow$ nižší kolize při souběžné aktualizaci indexu

- CREATE INDEX idx ON tab(attr) REVERSE;

Globální změny

- Vytvoření indexu
- Změna schématu
  - Viz další přednáška
- Rozdělení relací
  - Viz další přednáška