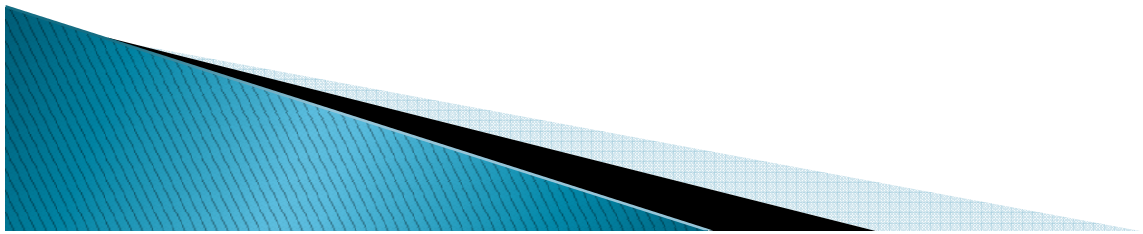
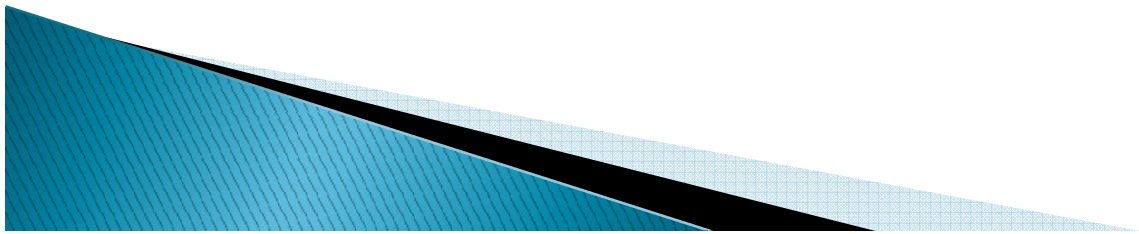
Databázové systémy 2

Optimalizace

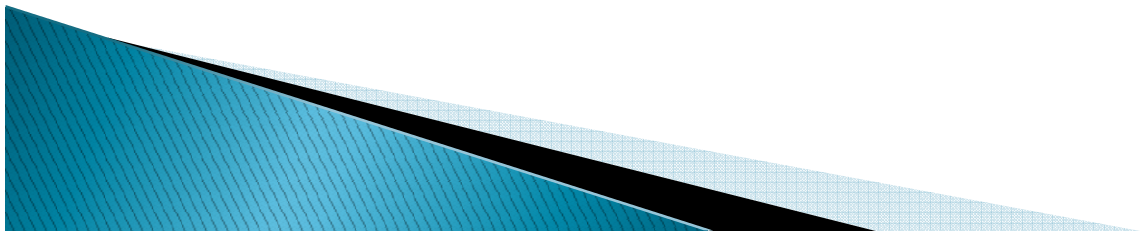


Optimalizace SQL dotazů



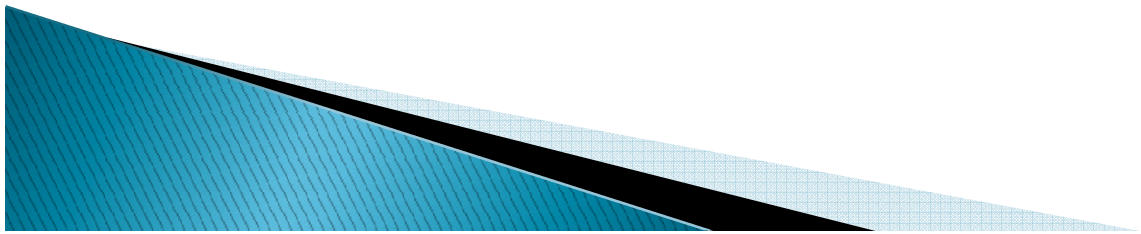
Motivace

- SQL je velmi flexibilní jazyk.
- Dvěma či více různými dotazy je možno obdržet stejná data.
- Rychlost různých dotazů ovšem nemusí být stejná i přesto, že vracejí stejná data.



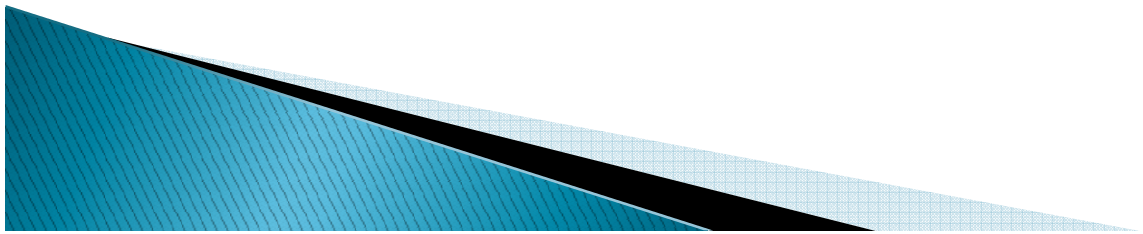
Obsah přednášky

- Proč optimalizujeme
- Obecná pravidla pro psaní SQL dotazů
- Oracle: zpracování SQL dotazů



Důvod optimalizace

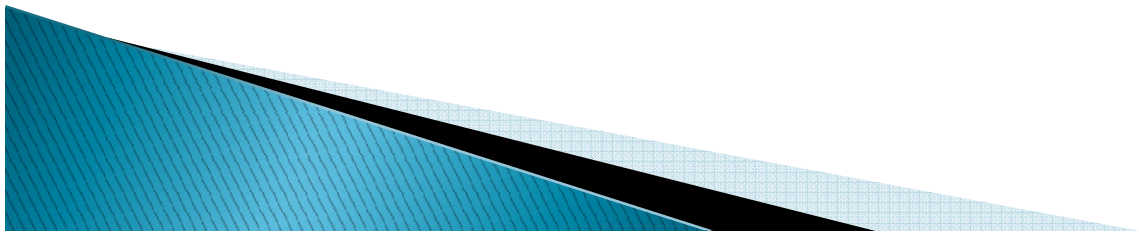
- Jedním z hlavních důvodů provádění optimalizace v databázových (DB) prostředcích je minimalizace nákladů.



Důvod optimalizace

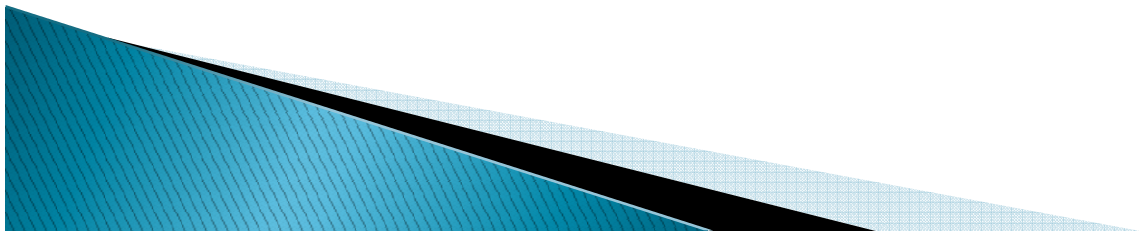
- Jedná se především o minimalizaci nákladů na:
  - zdrojový čas,
  - kapacitu paměti (prostor),
  - programátorskou práci.

  (= snažíme dosáhnout maximálního výkonu se stávajícími prostředky)



Kdo se na optimalizaci podílí

- Návrhář databáze (designer)
- Vývojář (developer)
- Správce databáze (DBA)
- Uživatel



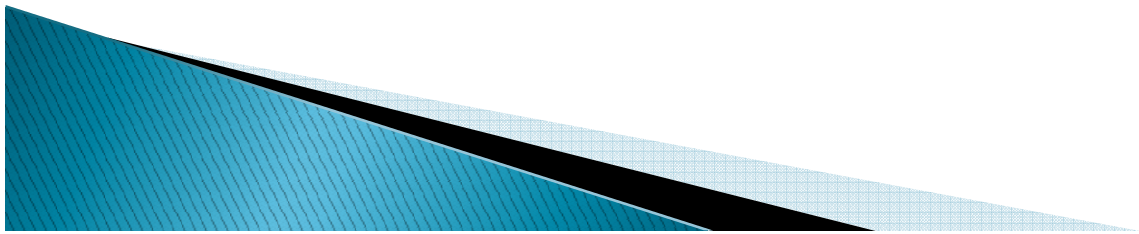
Kontext v SŘBD

- Jde o klíčový modul SŘBD
- cíl: učinit optimalizaci nezávislou na strategii zápisu dotazu

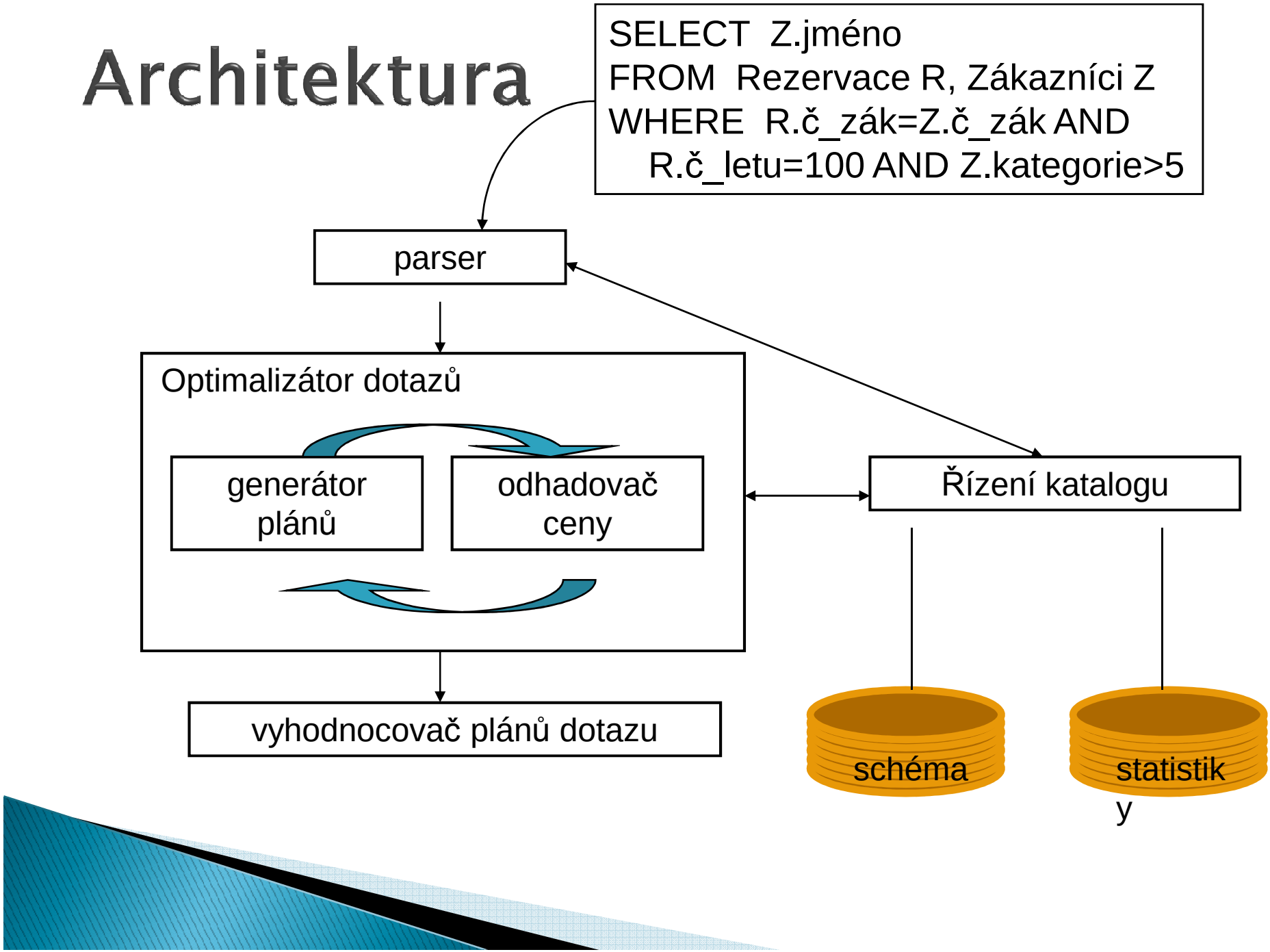
  protipříklady: navigační jazyky, interprety SQL
- paralela s vyhodnocováním aritmetických výrazů

  zde: časová složitost operací A_R pomocí I/O operací

  rozhodující: velikost relací, velikost aktivních domén, indexy, hašování, bitmapy atd.



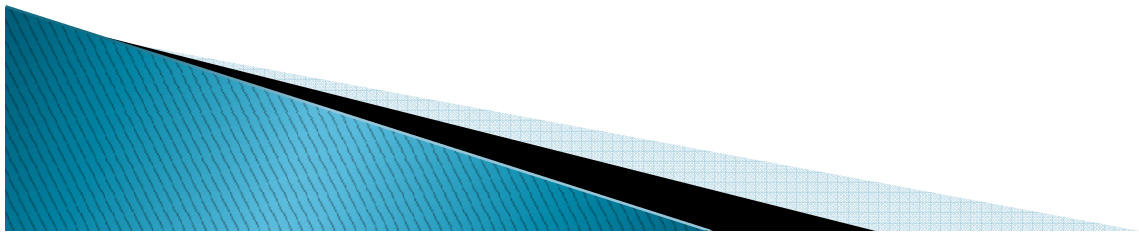
Architektura



Optimalizátor

- Fáze zpracování dotazu
  - převod do vnitřní formy
    - SQL $\rightarrow A_R$
    - lineární výraz $\rightarrow$ strom

    Poznámka: kalkul $\leftarrow \rightarrow A_R$ v polynomiálním čase v závislosti na délce výrazu
  - konverze do kanonického tvaru
  - optimalizace
  - plán vyhodnocení
  - generování kódu



Přehled problému

- *Plán vyhodnocení*: strom dotazu + algoritmus pro každou operaci.
- Dvě hlavní myšlenky:
  - jaké plány jsou uvažovány pro daný dotaz?
  - jak se odhaduje cena plánu?
- Z uvažovaných plánů se vybere ten s nejmenší cenou.

Př.: System R

- použití statistických dat pro odhad ceny,
- použití ekvivalentních algebraických výrazů,
- omezení na plány *doleva-do-hloubky*.

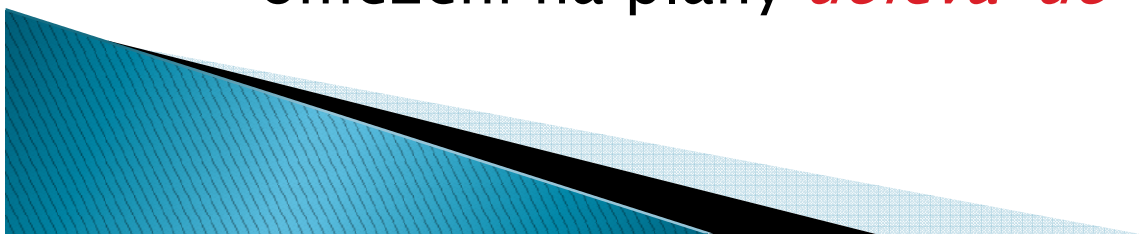


Schéma příkladu

Zákazníci (*č_zák*: int, *jméno*: string, *kategorie*: int, *věk*: real)
Rezervace(*č_zák*: int, *č_letu*: int, *datum*: date, *pozn*: string)

Sémantika: Zákazníci si rezervují lety do daného data.

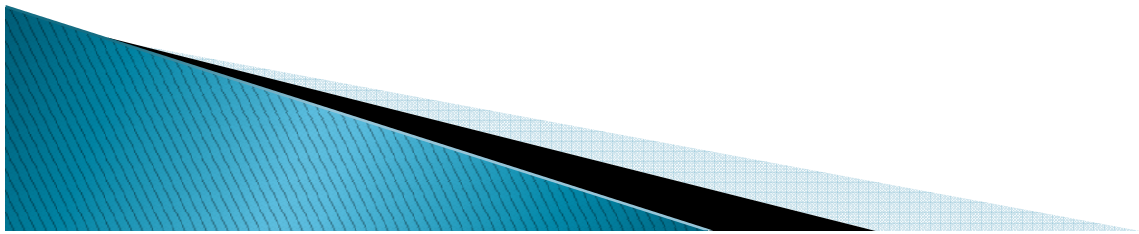
Parametry: B = 4 KByte

- Rezervace:

  R = 40 Byte, b = 100, p_R = 1000 stránek.

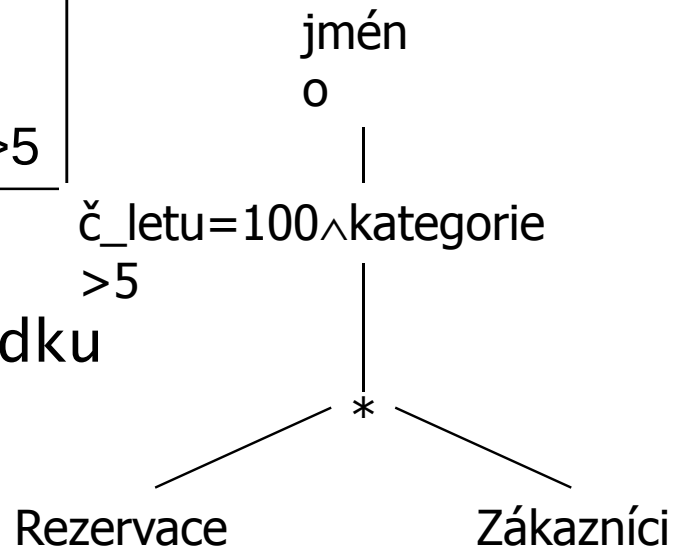
- Zákazníci:

  R = 50 Bytes, b = 80, p_Z = 500 stránek.



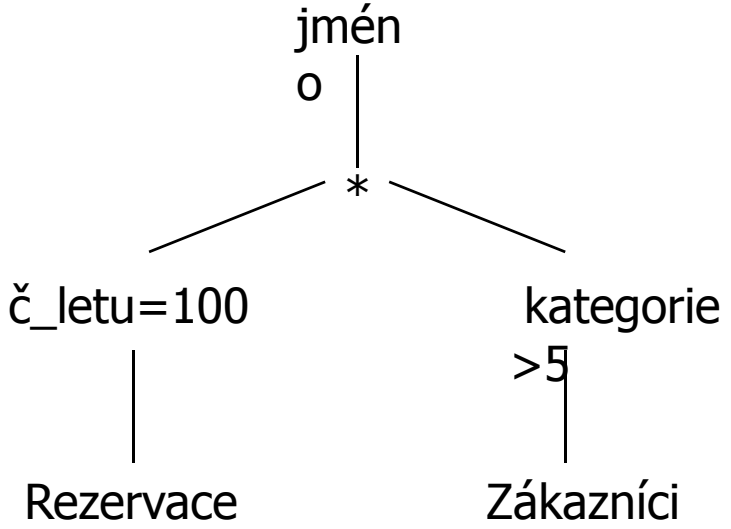
Alternativa 1

```
SELECT  Z.jméno
FROM  Rezervace R, Zákazníci Z
WHERE  R.č_zák=Z.č_zák AND
    R.č_letu=100 AND Z.kategorie>5
```



- Plán: spojení hnízděnými cykly, selekce+projekce při generování výsledku
- Cena: 500+500*1000 I/Os
- zřejmě nejhorší plán!
- Využít možnosti: selekce by měly být vyhodnoceny dříve, dostupné indexy, atd.
- Cíl optimalizace: Nalézt nejefektivnější plány, které vedou ke stejnému výsledku (odpovědi)

Alternativa 2 (bez indexů)



- Hlavní rozdíl: selekce nejdříve.
- Předpoklad: M=5 (5 bufferů). Výpočet ceny plánu:
  - Prohlídka Rezervace (1000) + write do T1 (10 stránek, máme-li 100 letů a rovnoměrné rozložení).
  - Prohlídka Zákazníci (500) + write do T2 (250 stránek, máme-li 10 kategorií rovnoměrné rozložení).
  - Sort(T1) (2*2*10), Sort(T2) (2*4*250), Merge(T1,T2) (10+250)
  - Součet: 1000+10+500+250+40+2000+260= 4060 I/O operací.

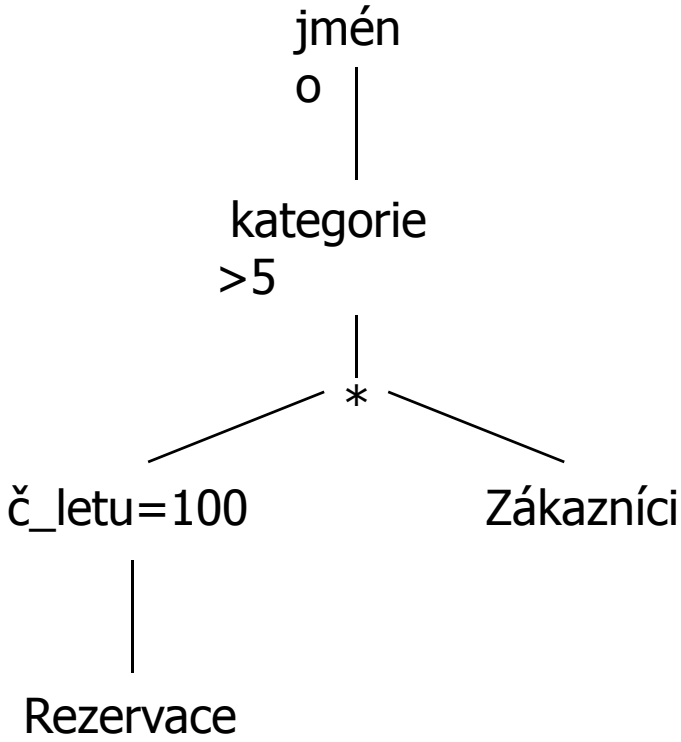
Pz.: třídění - n-cestným algoritmem třídění (T1 na 2 průchody, T2 na 4 průchody)

Zlepšení: projekce před tříděním - T1[č_zák], T2[č_zák,jméno]:

- T1 (1 stránka), T2 (166 stránek),
- Součet: 1000+1+500+166 + 2*1*1 + 2*4*166 + 1 + 167 = 3027 I/O operací.

Alternativa 3 (s indexy)

- s klastrovaným indexem *č_letu* v Rezervace, obdržíme 100,000/100 = 1000 n-tic na 1000/100 = 10 stránkách.



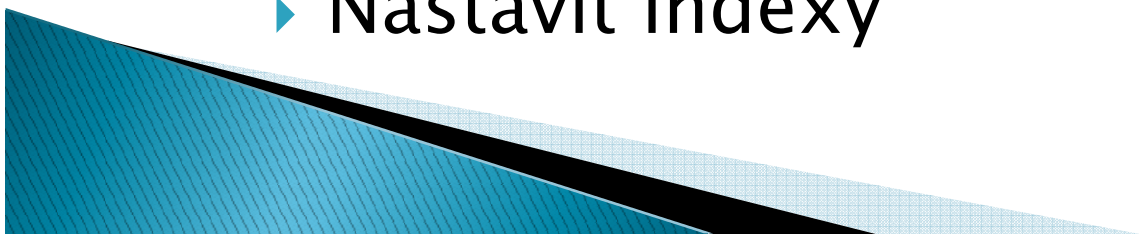
- Spojovací atribut je klíčem v Zákazníci
  - nejvýše jedna n-tice, neklastrovaný index na *č_zák* OK.
- Rozhodnutí nepropagovat *kategorie >5* před spojení je založena na dostupnosti indexu *č_zák* v tabulce Zákazníci.
- Cena: čtení stránek z Rezervace (10); pro každou rezervační n-tici se čte 1 stránka ze Zákazníci (1000 ×);

  součet: 1010 I/O operací

Obecná pravidla pro každého

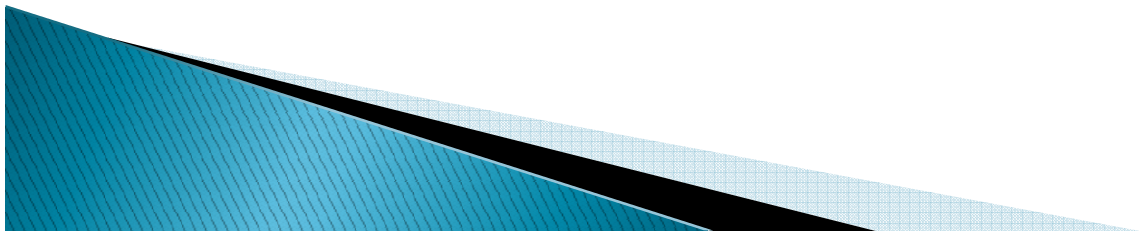
Obecná pravidla pro psaní SQL dotazů

- Vyjmenovat sloupce
- Používat co nejméně klauzuli LIKE
- Používat co nejméně klauzule IN, NOT IN
- Používat klauzule typu LIMIT
- Na začátek dávat obecnější podmínky
- Výběr vhodného pořadí spojení
- Používat hinty
- Nastavit indexy



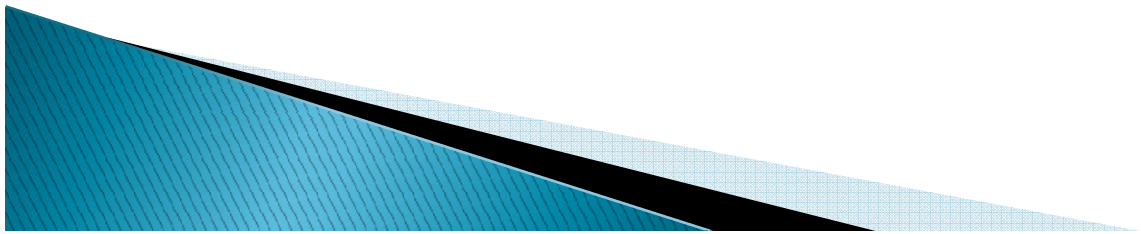
Pojmenování sloupců

- V *SELECT* dotazech nepoužívat v seznamu sloupců hvězdičku *(*)*
- Ve většině případů nepracujeme se všemi sloupci výsledku
- *SELECT * FROM Lide*
- *SELECT Jmeno, Prijmeni FROM Lide*



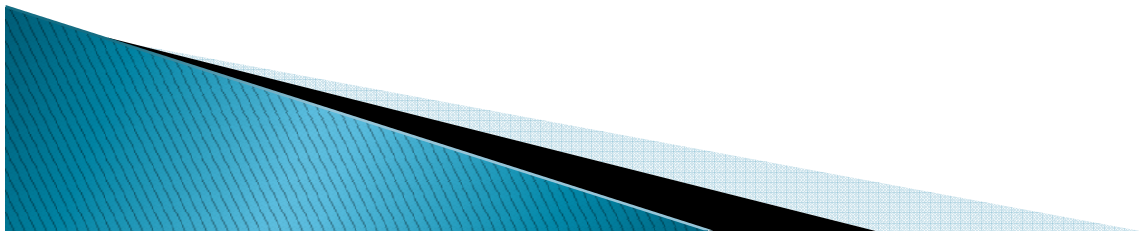
Pojmenování sloupců

- Používáte-li v *SELECT* dotazu všechny sloupce, používejte také výpis jednotlivých sloupců
- Databáze nemusí zjišťovat seznam sloupců tabulky



Problematické použití LIKE

- Nedoporučuje se používat pro vyhledávání ve velkých textových polích (můžou obsahovat až několik GB textu)
- Zamyslet se, zda nejde vyhledávání provést jinou metodou

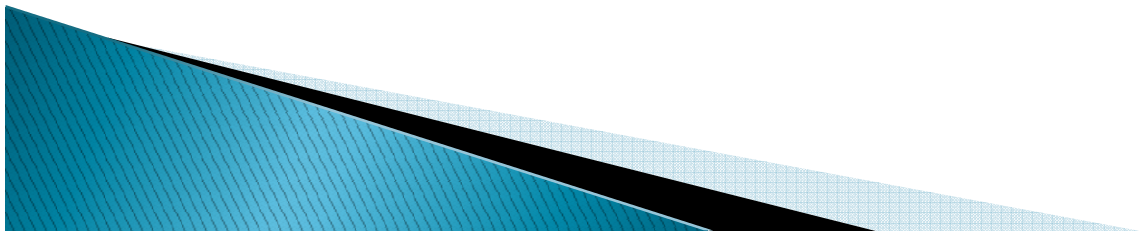


Používat co nejméně klauzuli IN, NOT IN

- *Vhodnější je použití příkazů WHERE a WHERE NOT EXISTS*

... WHERE Doprava IN ('Ford', 'Octavia', 'Seat', 'Peugeot');

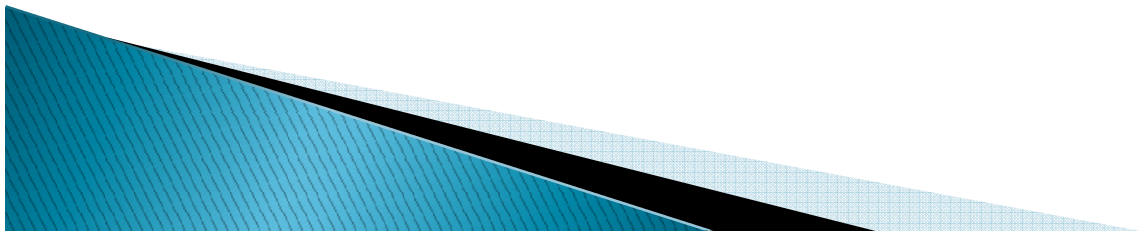
... WHERE Typ_Dopravy = 'Automobil';



Klauzule typu LIMIT

- V případech, kdy vybíráme např. nejstaršího člověka, můžeme použít dotaz:

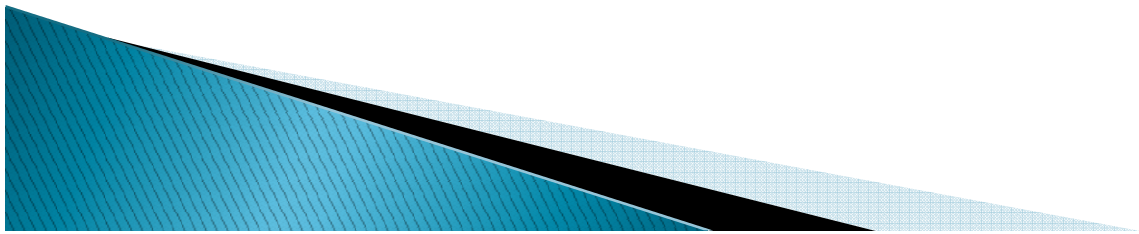
  SELECT Jmeno, Prijmeni FROM Lide ORDER BY Vek DESC



Klauzule typu LIMIT

- Lepší řešení:

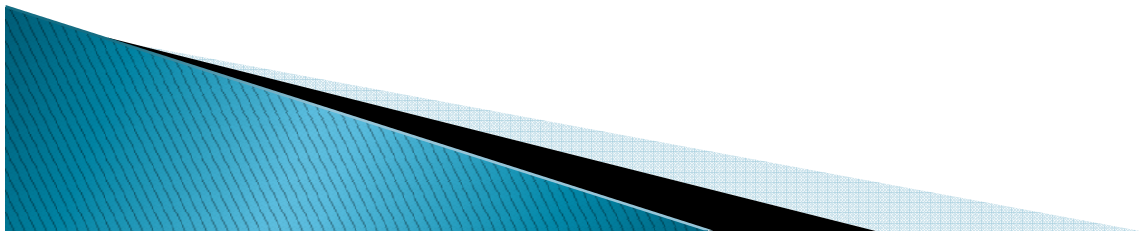
SELECT Jmeno, Prijmeni FROM Lide ORDER BY Vek DESC LIMIT 0,1



Důležitá pravidla

- V klauzuli *WHERE* dávat na začátek podmínky, po kterých vypadne ze seznamu nejvíce záznamů

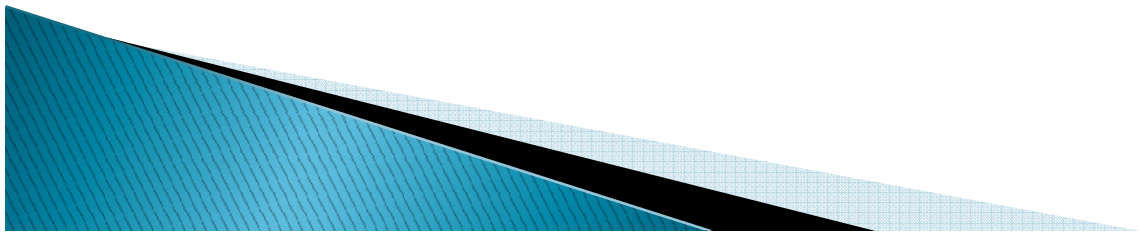
:–/



Důležitá pravidla

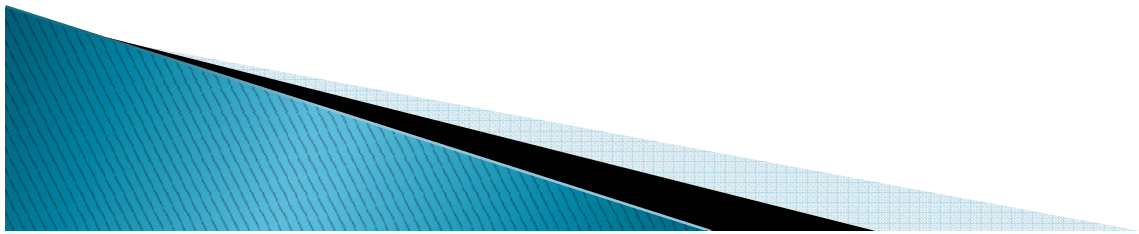
- Příklad: V tabulce *Lide* hledáme ženy starší 18 let

  SELECT Jmeno, Prijmeni FROM Lide WHERE Pohlavi = 'Z' AND Vek > 18



Důležitá pravidla

- DS nejprve vyhledá záznamy, vyhovující první podmínce, z nich pak vybírá záznamy vyhovující druhé podmínce
- Snažíme se, aby systém vyřadil na začátku co nejvíce řádků; ty se pak již při další podmínce nezkoumají…

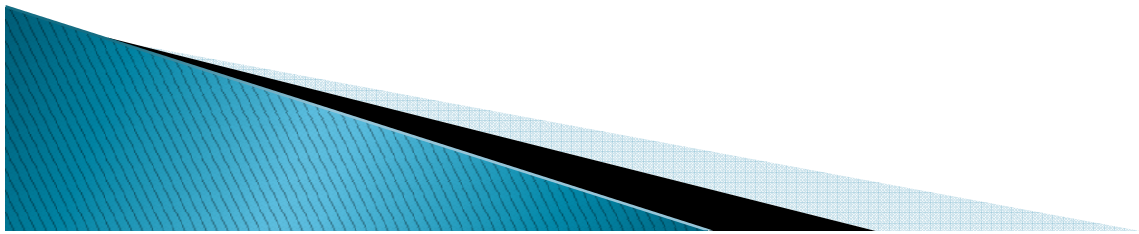


Výběr vhodného pořadí spojení

1) vyhnout se plnému prohledávání tabulky (pokud možno využít index)

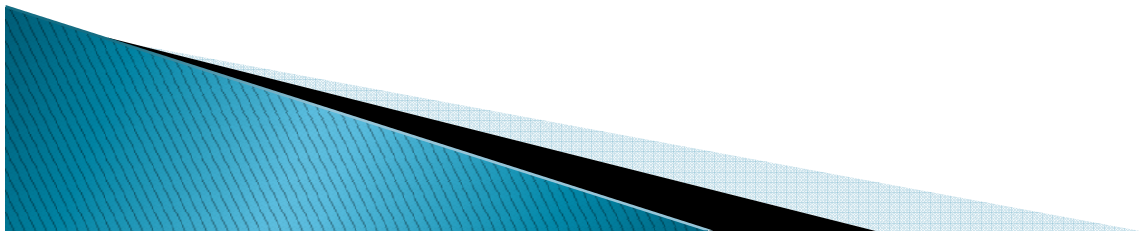
2) efektivně vybírat takové indexy, které načtou z tabulky co nejméně záznamů

3) vybrat takové pořadí spojení ze všech možných pořadí, aby bylo spojeno co nejméně položek



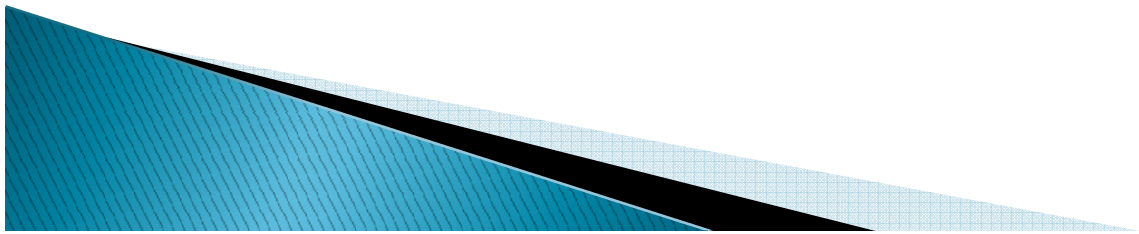
Důležitá pravidla – souhrn

- Použití UNION ALL místo UNION
- Spojování tabulek s využitím indexů
- Vytváření indexů pro atributy podle nichž se třídí v klauzili ORDER BY
- Provádění analýzy na indexovaných sloupcích



Použití hintů

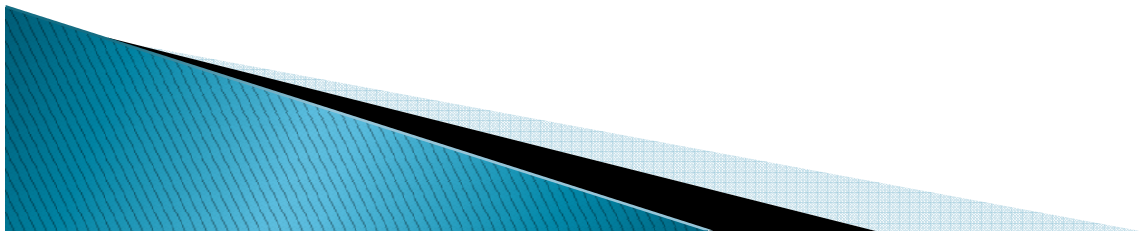
- Hint = podnět, kterým optimalizátoru určíme, jaký má použít plán vykonávání dotazu
- Hinty se aplikují na blok dotazu, ve kterém se vyskytují.



Použití hintů

- SELECT jmeno, prijmeni, plat FROM ucitel WHERE pohlavi='M';

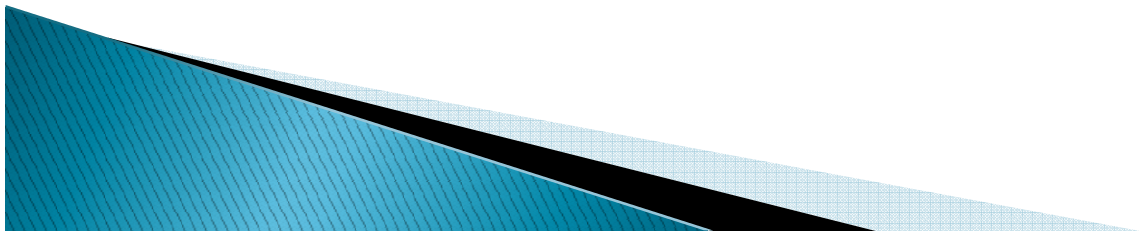
- Optimalizátor by v takovémto případě zřejmě zvolil full table scan, protože pohlaví může obsahovat pouze dvě hodnoty, tedy vrácených řádků by měla být velká část ze všech možných.



Použití hintů

- Pokud však víme, že učitelů – mužů je hodně málo (například ukládáme pouze učitele z mateřských školek), pak si můžeme pomocí hintu vynutit rychlejší přístup – index scan.

- SELECT /*+ INDEX(ucitel pohlavi_index) */ jmeno, prijmeni, plat FROM ucitele WHERE pohlavi='M';



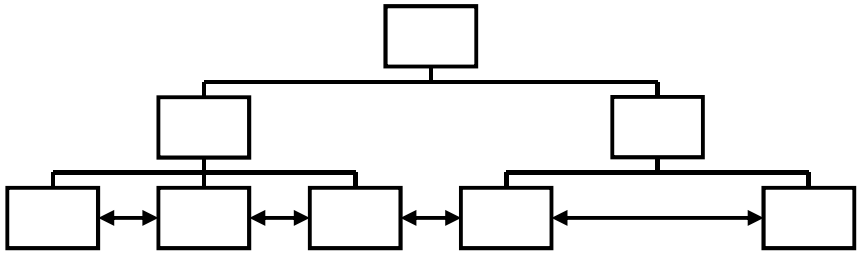
Indexy

- Procházení tabulky pomocí indexu trvá mnohem kratší dobu než procházení tabulky bez jeho použití.
- Změna indexů se zdá být nejlepším řešením pro optimalizaci, jelikož má větší sílu než změna SQL dotazu či změna dat.
- Samotné vytvoření indexů však nelze brát v úvahu jako univerzální řešení problému.

Indexy

- Tvorba indexů není v SQL-92 standardizována
- Jednotlivé databázové systémy řeší tvorbu indexů svými prostředky,
  které jsou navzájem více či méně podobné
  - Může se lišit
    - syntaxe
    - podpora různých typů indexů
    - jejich (ne)použití pro daný dotaz a obsah tabulky

B-tree indexy



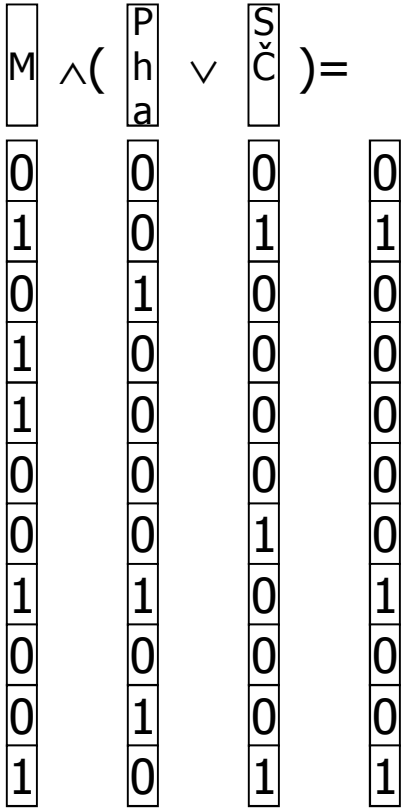
- Obvykle redundantní B+ stromy
  - Hodnoty v listech
  - Listy oboustranně linkované
    pro snadný sekvenční průchod.
  - Vhodné pro sloupce s vysokou selektivitou
    (počtem různých hodnot ve sloupci).
  - Vícesloupcové (složené) indexy mohou zvýšit selektivitu.
    - Použitelné, pokud dotaz omezuje hodnoty prvních *k* sloupců indexu,
      přičemž prvních *k*–1 sloupců musí být omezeno na rovnost
    - Nepoužitelné, pokud v dotazu není omezení na první sloupec indexu
  - Nad jednou tabulkou v jednom dotazu nelze obvykle kombinovat
    více B-tree indexů. Dotaz se vyhodnocuje s použitím jednoho z indexů a ostatní podmínky se dopočítávají

Bitmapové indexy

- Pro každou hodnotu sloupce / výrazu vytvořen binární řetězec obsahující 1 právě pro řádky s danou hodnotou
  - Vhodné pro sloupce s nízkou selektivitou
  - Lze kombinovat více bitmapových indexů nad jednou tabulkou pro zvýšení selektivity
  - Kombinací více bitmap se zvyšuje selektivita indexu

```
SELECT * FROM Obyvatel
WHERE Pohlaví='M'
   AND (Kraj='Pha' OR Kraj='SČ');
```

  - Kombinace tří bitmapových řetězců



Kdy indexy (ne)pomohou

- Nepomohou
  - Pokud je procento vyhovujících záznamů velké
    - zvýšená režie s přístupem k řádkům v nesekvenčním pořadí, daném indexem
  - Při dotazech na hodnotu NULL
    - v indexech se běžně neukládají
- Pomohou
  - V dotazech na rovnost sloupce s konstantou
  - V dotazech na náležení hodnoty do intervalu

Kdy (ne)vytvářet indexy

- Jak Oracle, tak řada dalších databázových systémů vytváří automaticky unikátní indexy pro
  - primární klíče
    - jméno bývá shodné se jménem omezení
  - kandidátní klíče (UNIQUE sloupce)
    - jméno bývá shodné se jménem omezení

Kdy (ne)vytvářet indexy

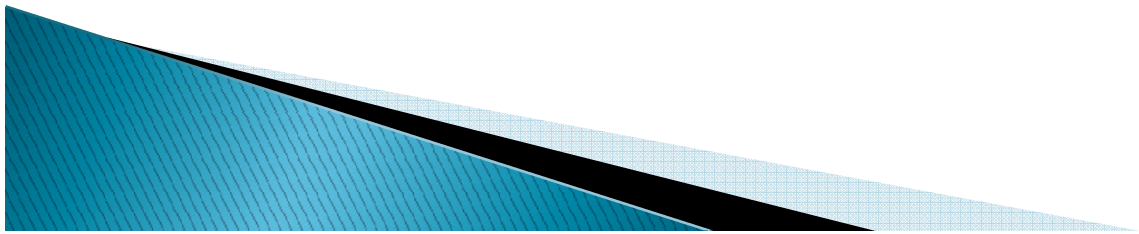
- Důležité je vždy vytvářet indexy pro cizí klíče !!!
  - Zrychlení odezvy při manipulaci s nadřízenou tabulkou
    - Při rušení nadřízené řádky je bez indexu nutné projít celou podřízenou tabulku, zda neobsahuje závislé řádky
    - Pokud je aktivované kaskádové mazání a odkazy jsou víceúrovňové, pro každou nalezenou podřízenou řádku je nutné projít celou její pořízenou tabulku atd. atd.
  - Průchod tabulkou čte i bloky, obsahující zrušené záznamy v tabulce
  - Průchod přes index najde efektivně všechny existující závislé řádky bez nutnosti čtení samotné tabulky
  - Např. Oracle řeší nemožnost efektivně najít a zamknout podřízené řádky zamknutím celé podřízené tabulky, čímž velmi omezuje možnost běhu aplikace s více uživateli.

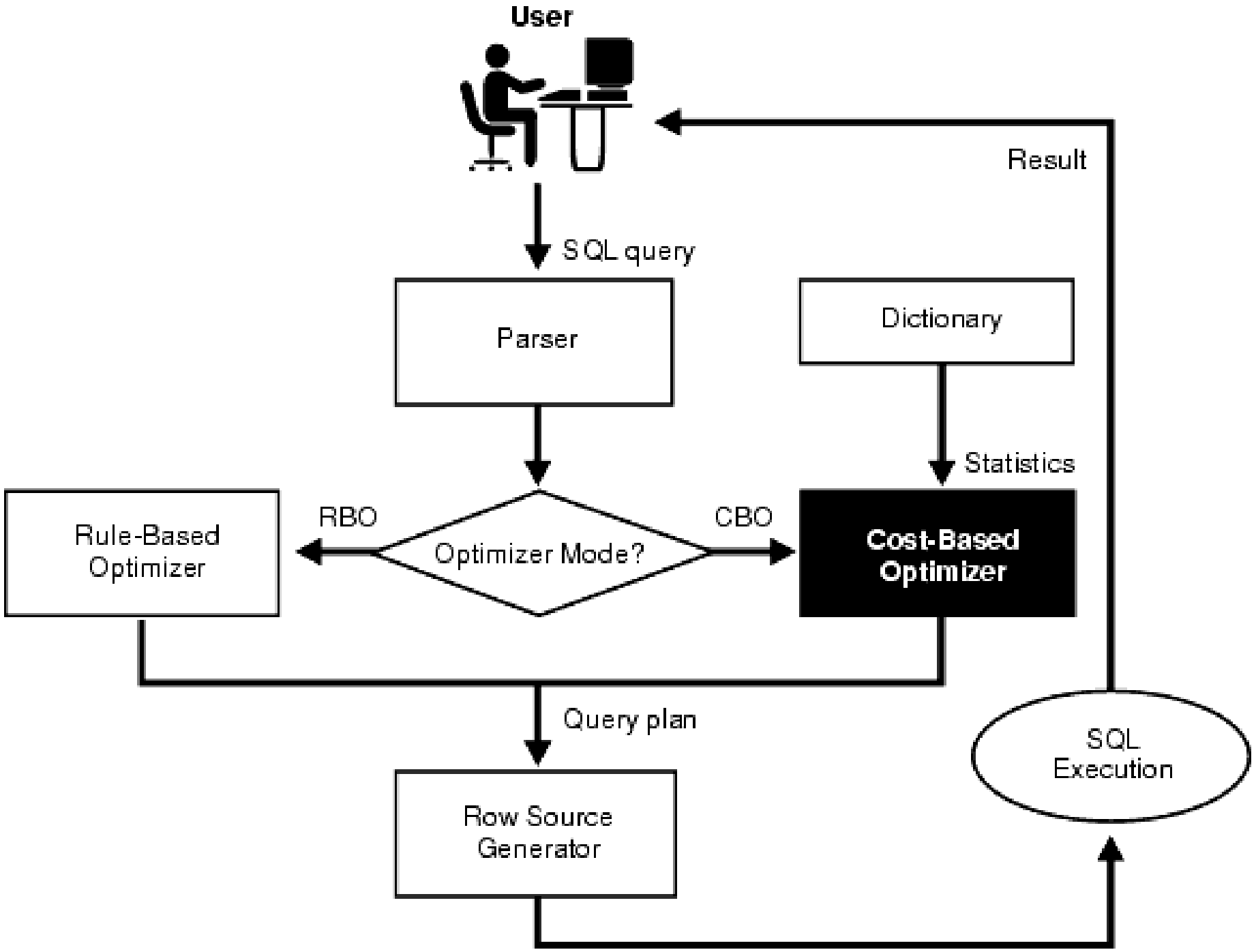
Kdy (ne)vytvářet indexy

- Indexy by se jinak měly vytvářet jen v případě, pokud výrazně pomohou často kladeným dotazům
  - V opačném případě spíše zdržují aktualizační operace

Oracle: Zpracování SQL dotazů

- Zpracování SQL příkazů se sestává z následujících komponent:
  - Parser
  - Optimalizátor
  - Generátor řádkových zdrojů (row source generator)
  - Vlastní provádění (SQL execution)



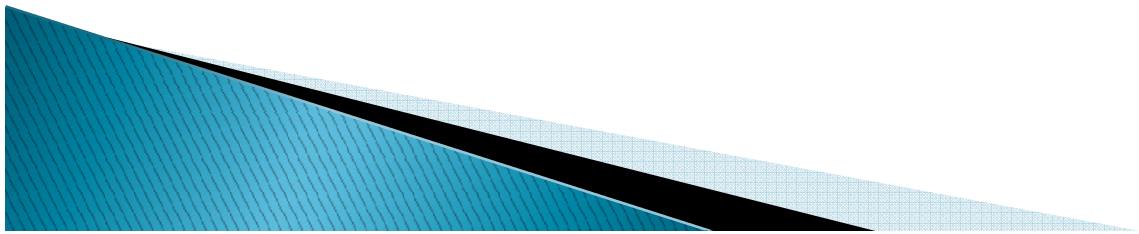

User
Result
SQL query
Parser
Dictionary
Statistics
RBO
CBO
Rule-Based Optimizer
Optimizer Mode?
Cost-Based Optimizer
Query plan
SQL Execution
Row Source Generator

Optimalizátor

- Jádro celého zpracování
- Analyzuje sémantiku dotazu
- Hledá optimální způsob jeho provádění
- V Oracle rozeznáváme:
  - Rule-based optimizer (RBO)
  - Cost-based optimizer (CBO)
- Liší se v přístupu, jakým hledají optimální plán vykonávání

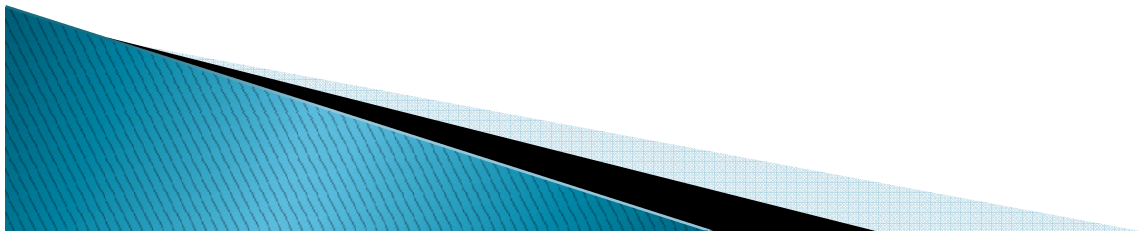
Optimalizátor

- **rule-based optimizer** vyhodnocuje jednotlivé přístupové cesty pomocí předem daného systému pravidel
- **cost-based optimizer** hledá plán s nejmenšími "náklady" (využívá statistiky)
- Oracle doporučuje používat pouze CBO



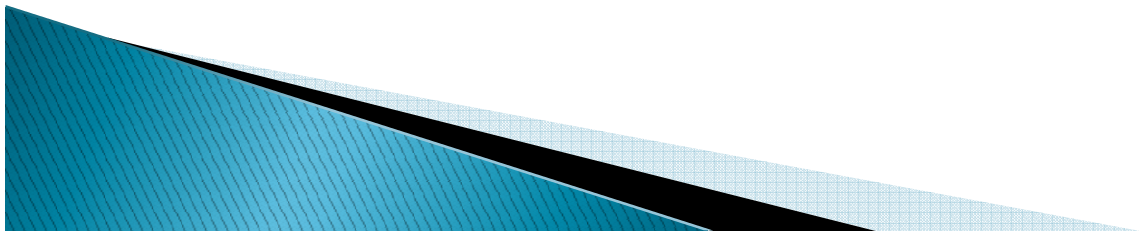
Optimalizátor

- Výstupem optimalizátoru je plán vykonávání (execution plan), který určuje:
  - přístupové cesty k jednotlivým tabulkám používaným dotazem,
  - pořadí jejich spojování (join order).



Statistiky

- Statistiky tvoří celá řada údajů o databázových objektech (tabulkách, indexech)
- Některé z těchto údajů jsou přístupné prostřednictvím tabulek a pohledů slovníku dat a může je tedy využívat i uživatel databáze.
- Aktualizují se výpočtem nebo odhadem.

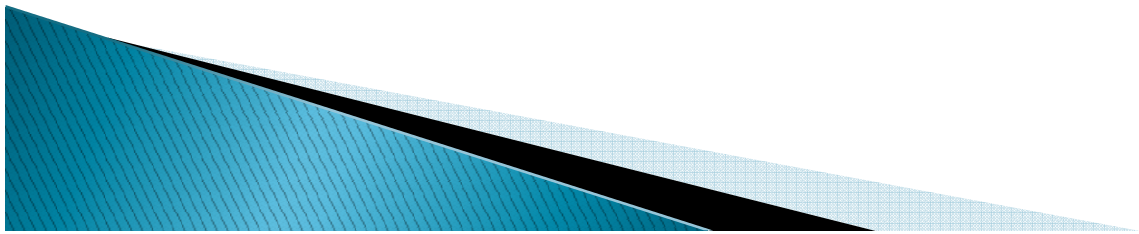


Statistiky

- **údaje o tabulkách** (počet řádků, počet bloků, počet nevyužitých bloků, průměrnou délku záznamu)
- **údaje o sloupcích** (počet unikátních hodnot, počet prázdných (NULL) hodnot, histogram popisující distribuci dat)
- **údaje o indexech** (počet listových bloků, počet úrovní, clusterovací faktor)

Možnosti ladění

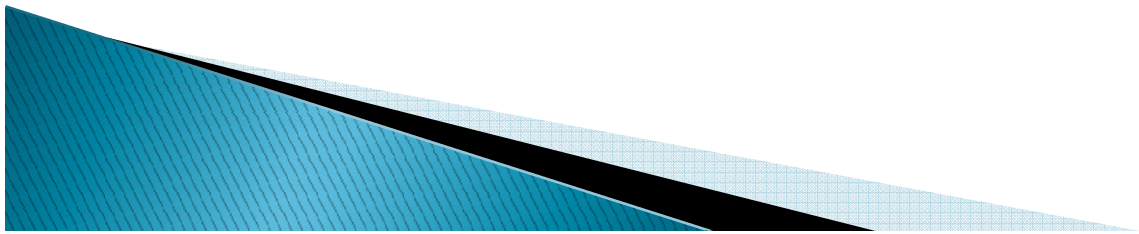
- OPTIMIZER_MODE – pro dosažení maximální propustnosti (s co nejmenším využitím zdrojů), nebo dosažení co nejlepší odezvy (co nejdříve vrátit první výsledky)
- SORT_AREA_SIZE – Určuje velikost paměti využívané při třídění a nepřímou úměrou ovlivňuje cenu spojení



Možnosti ladění

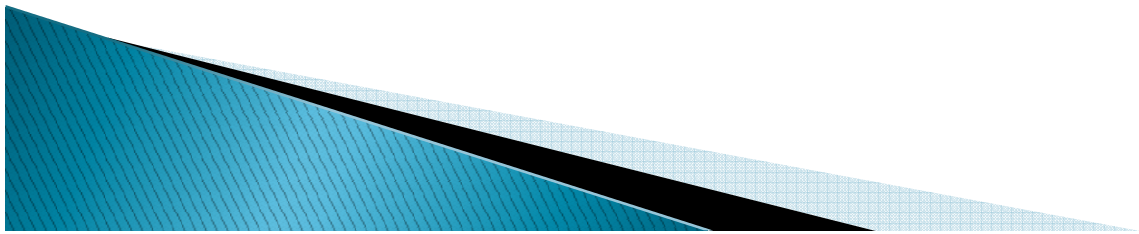
- CURSOR_SHARING – Tento parametr určuje, zda se bude dotaz vyhodnocovat přesně jak byl zadán nebo se literály nahradí vázanými proměnnými

- HASH_AREA_SIZE – Určuje velikost paměti využívané při hašovaném spojování a nepřímou úměrou ovlivňuje cenu hašovaného spojení

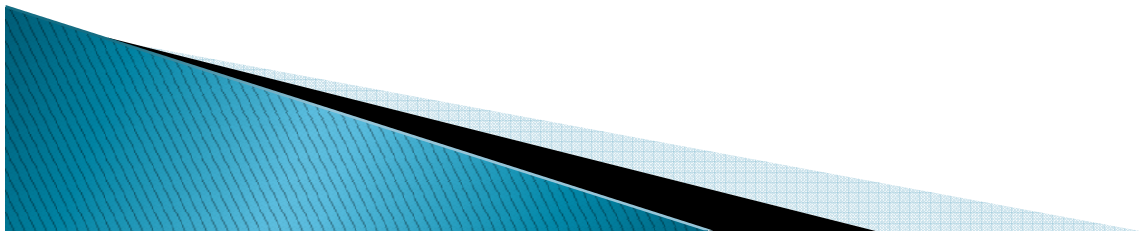


Minimalizace reparsingu dotazů

- Toho dosáhneme používáním jednotného zápisu dotazů a používáním vazebních proměnných místo konstant

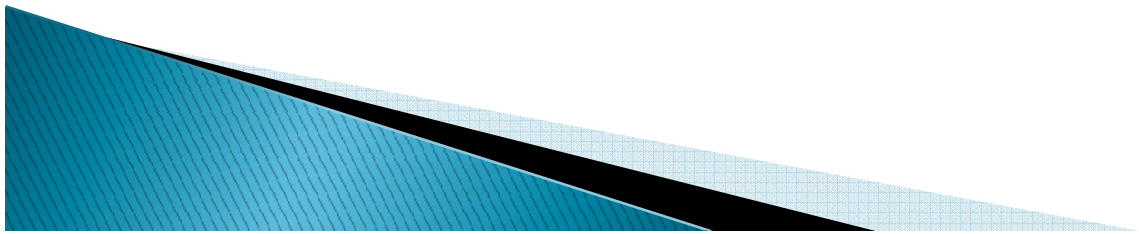


„Dobrý návrh databáze a aplikace má daleko větší vliv na výkon, než sebelepší nastavení parametrů instance.“



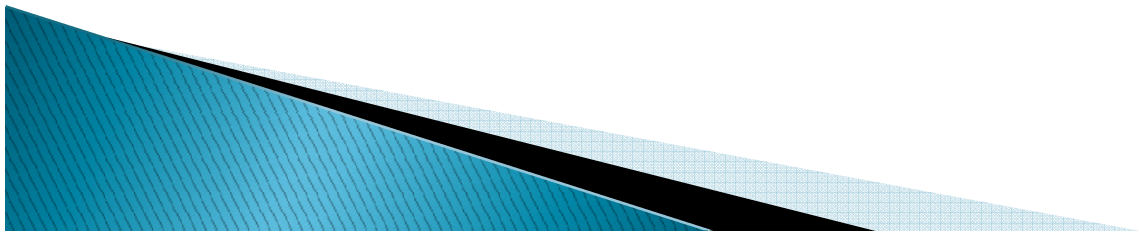
Odkazy na zdroje dat

http://www.oracle.com
http://www.pcsvet.cz/art/article.php?id=197
http://www.dbs-intro.com/dbplus/ch01.html
http://www.sweb.cz/nidrle.vaclav/oracle2/optimaliza
ce.html



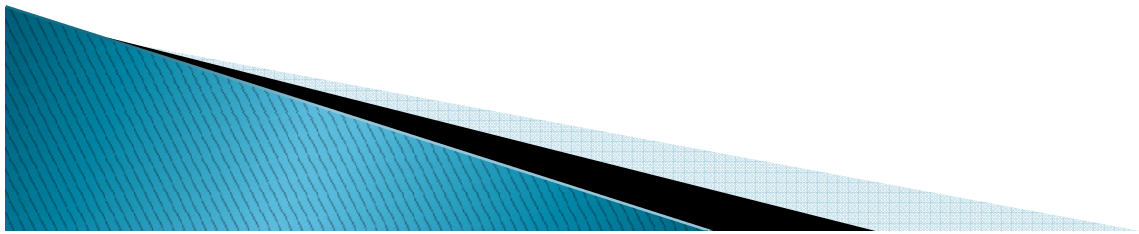
Optimalizace SQL dotazů

Plány provedení dotazu
Ovlivnění optimalizátoru



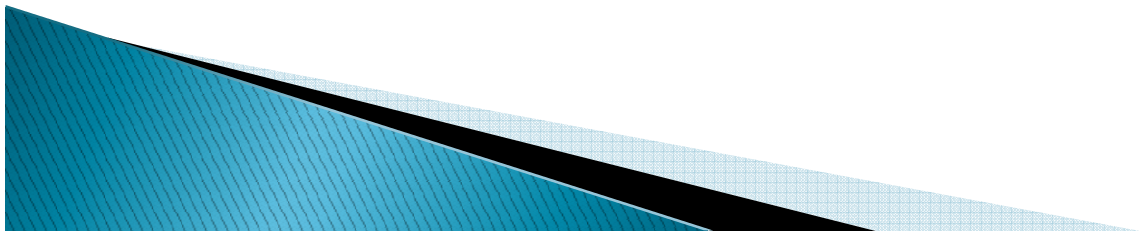
Optimalizace dotazů

- Jeden dotaz lze napsat více způsoby
  - Shodná sémantika
  - Rozdílný způsob výpočtu výsledku
  - Doba výpočtu se může lišit i řádově !!
- O způsobu výpočtu rozhoduje optimalizátor uvnitř databázového serveru
- Potřebné je
  - Umět zjistit, jakým způsobem výpočet proběhne
  - Zvolit nejlépe optimalizovatelný zápis dotazu nebo optimalizátoru pomoci s výběrem



Typy optimalizace

- V Oracle
  - Starší RULE BASED optimalizace (RBO)
    - Odvozuje plán ze syntaxe příkazu a existence indexů
  - Novější COST BASED optimalizace (CBO)
    - Oracle 8+, doporučována pro lepší vlastnosti
    - Založena na statistikách, počítá cenu zdrojů provedení operace (čas, prostor, třídící operace, …)
    - Dokáže rozlišit plány i pro různé hodnoty konstant v dotazu

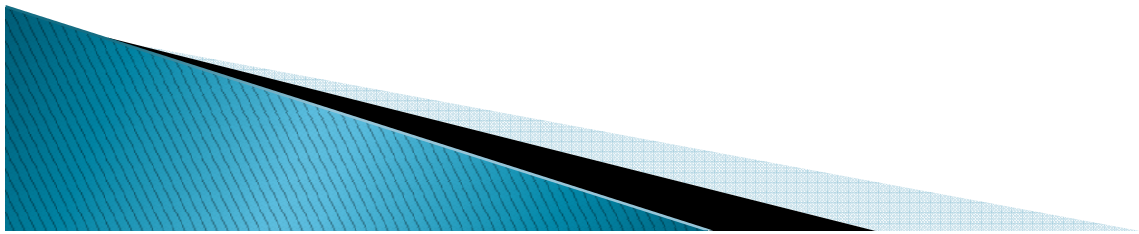


Rule-Based Optimalizace dotazů

- Cena přístupu k podmnožině řádek v tabulce v klesajícím pořadí
  - Full-scan
    - Prochází se celá tabulka, u každé řádky se ověří podmínka
    - Může být vhodné, pokud procento vyhovujících řádek je dost velké
  - Index-Range-Scan
    - Vyhledání intervalu v indexu, ověření ostatních podmínek v odkazovaných řádcích
  - Unique-Index-Scan
    - Vyhledání jediné možné vyhovující řádky podle unikátního indexu
  - ROWID-Scan
    - Vyhledání řádky na základě známé hodnoty jejího fyzického identifikátoru v databázi

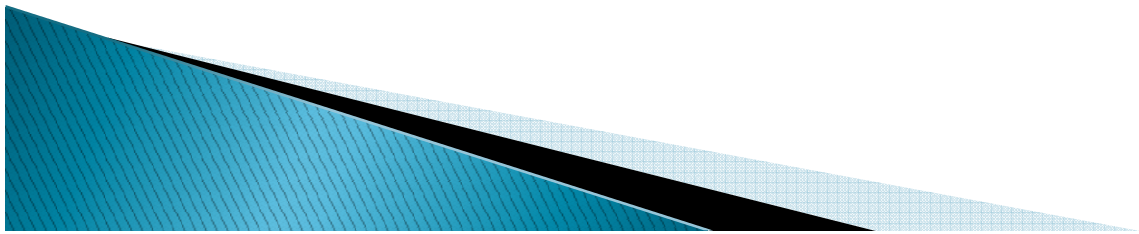
Optimalizace dotazů

- Cena operace JOIN dvou tabulek
  - Databáze se snaží zvolit tabulku s dražším přístupem jako hlavní tabulku
  - Ke každé nalezené vyhovující řádce dohledává odpovídající řádky ve druhé tabulce
  - Pokud obě tabulky nabízí pouze Full-Scan, data obou se setřídí a provede se Merge-Join



Optimalizace dotazů

- Jak zjistit způsob provedení příkazu?
  - Nejprve je nutné mít tabulku PLAN_TABLE s odpovídající strukturou, do které plánovač ukládá informace o plánu provedení příkazu
    `@?\rdbms\admin\utlxplan[.sql]`
  - SQL*Plus nabízí přepínač
    `SET AUTOTRACE {OFF|ON|TRACEONLY}`
  - Oracle obsahuje příkaz `EXPLAIN PLAN`



Tabulka PLAN_TABLE

- Zajímavé sloupce

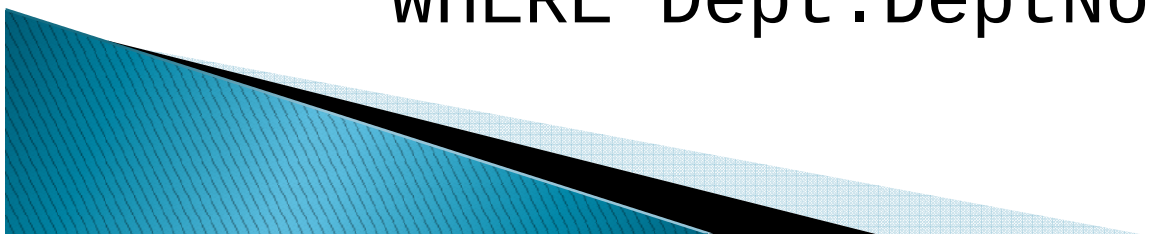
STATEMENT_ID	VARCHAR2(30)
OPERATION	VARCHAR2(30)
OPTIONS	VARCHAR2(30)
OBJECT_OWNER	VARCHAR2(30)
OBJECT_NAME	VARCHAR2(30)
OBJECT_TYPE	VARCHAR2(30)
ID	NUMBER(38)
PARENT_ID	NUMBER(38)
COST	NUMBER(38)
…	

- Identifikátor příkazu, shodný pro všechny řádky plánu
- Operace (SCAN, SORT, …)
- Upřesnění operace (FULL, RANGE, UNIQUE, …)
- Vlastník zdroje
- Jméno zdroje
- Typ zroje (TABLE, INDEX, CLUSTER, …)
- Identifikace kroku v plánu s daným STATEMENT_ID
- Identifikace rodičovského kroku ve stromu provedení
- Odhadovaná cena
- …

- Strom plánu se čte od listů ke kořeni
  - Rodičovská operace se provádí po dokončení operací v potomcích
  - Kořenová operace má ID=0

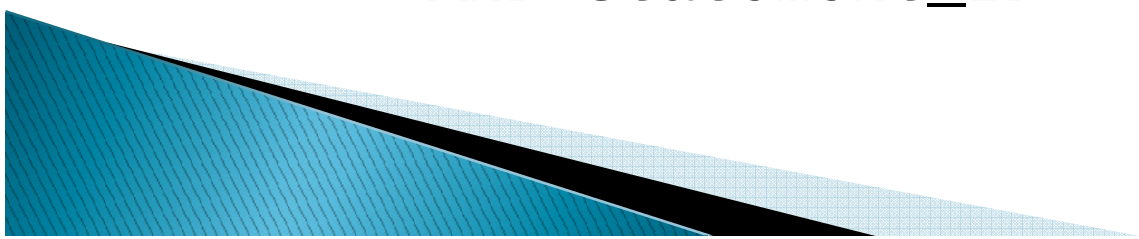
Příkaz EXPLAIN PLAN

- **EXPLAIN PLAN**
  SET STATEMENT_ID = ’jméno’
  [INTO tabulka]
  FOR příkaz;
- EXPLAIN PLAN
  SET STATEMENT_ID = ’emp_dept’
  FOR
  SELECT Emp.*, Dept.Loc
  FROM Dept, Emp
  WHERE Dept.DeptNo = Emp.Deptno;



Příkaz EXPLAIN PLAN

- Získání plánu provedení např. příkazem
-
```
SELECT
 LPad(' ',2*Level-1)||operation
 ||' '||options
 ||' '||object_name AS text
FROM Plan_Table
START WITH Statement_ID='příkaz'
 AND ID = 0
CONNECT BY Parent_ID = PRIOR ID
 AND Statement_ID = PRIOR Statement_ID;
```



Jak psát optimalizovatelné dotazy

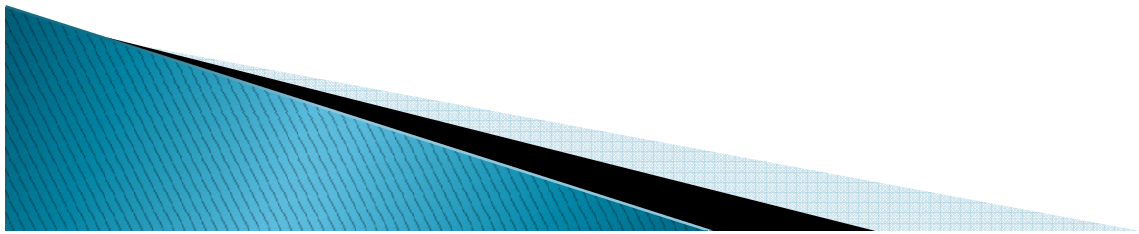
- V aplikacích používat místo konstant tzv. placeholdery a aplikační proměnné
  - Dva různé dotazy mají dva samostatné plány provedení. Jejich vytvoření zabírá čas a jiné zdroje databáze
    - SELECT * FROM Emp WHERE DeptNo=10;
      SELECT * FROM Emp WHERE DeptNo=20;
    - SELECT * FROM Emp WHERE DeptNo=:d;
  - Někdy ovšem mohou dva plány pomoci, pokud se princip provedení odůvodněně liší.
    - SELECT * FROM Vojaci WHERE Pohlavi='M'
    - SELECT * FROM Vojaci WHERE Pohlavi='Ž'

90% dat (full s.)

10% dat (range s.)

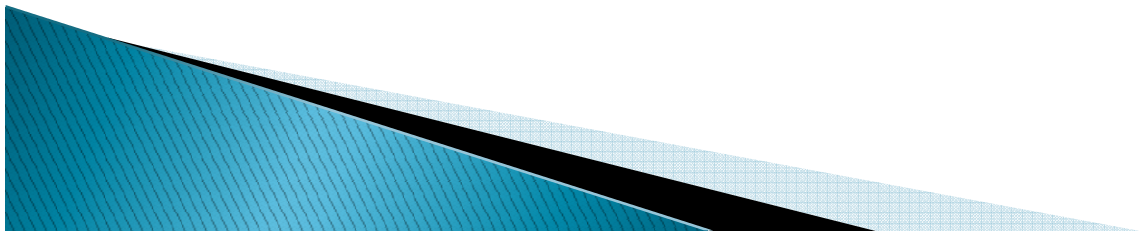
Jak psát optimalizovatelné dotazy

- Jeden dotaz psát všude přesně stejně
  - Různé zápisy databáze vždy znovu analyzuje a vymýšlí plán provedení
    - SELECT * FROM Emp
      WHERE Ename LIKE 'A%' AND DeptNo=10;
    - SELECT * FROM Emp
      WHERE DeptNo=10 AND Ename LIKE 'A%';



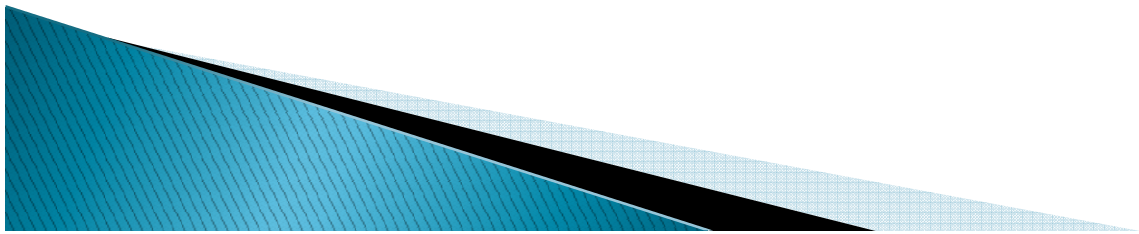
Nevýhody rule-based optimalizace

- Pokud existuje více neunikátních indexů na jedné tabulce, nemusí optimalizátor vybrat ten nejlepší
- SELECT * FROM Osoba
  WHERE Jmeno='Jan' AND Mesto='Praha';
  - Buďto se přes index hledají Janové, a ověřuje se bydliště, nebo se hledají Pražáci a ověřuje se jméno



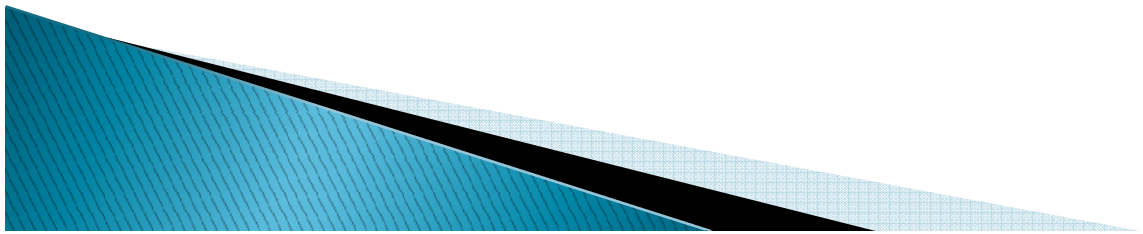
Nevýhody rule-based optimalizace

- Pokud existuje více neunikátních indexů na jedné tabulce, nemusí optimalizátor vybrat ten nejlepší
- Použití určitého indexu je možné optimalizátoru znemožnit použitím výrazu v dotazu
- SELECT * FROM Osoba
  WHERE **Jmeno**||''='Jan' AND …;
  - Index přes jméno nelze použít, použije se město



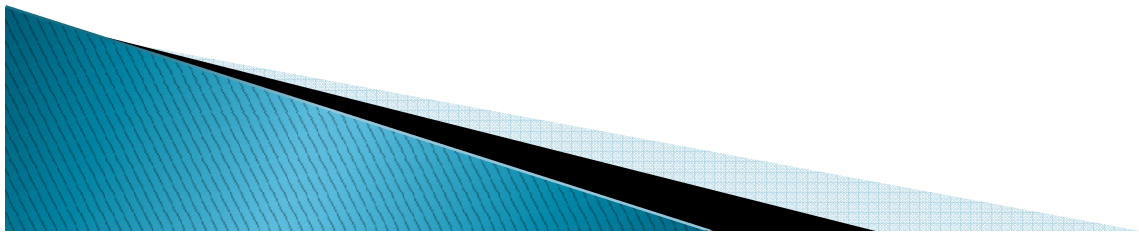
Cost-based optimalizace

- Pro jednotlivé plány se počítá cena provedení v řadě hledisek
  - Množství I/O operací, řádek, Byte, …
  - Cena prováděných třídících operací
  - Cena za HASH operace
- Vybírá se plán s nejnižší váženou cenou



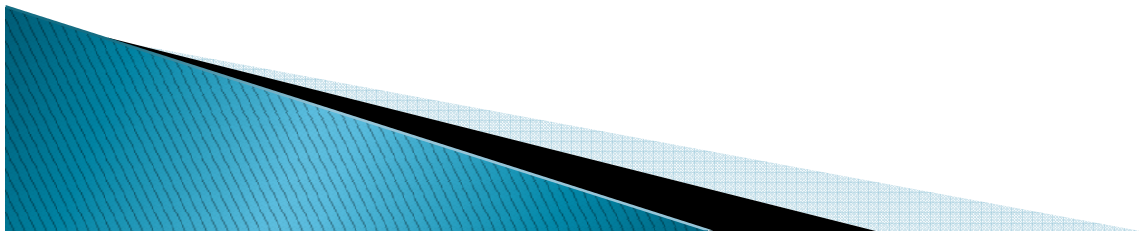
Cost-based optimalizace

- Využívá statistických informací o datech
  - Počet různých hodnot ve sloupci,
    Histogramy rozložení hodnot ve sloupci,
    Počet řádek v tabulce,
    Průměrná délka jedné řádky
    - Pro konkrétní hodnotu nebo interval hodnot lze odhadnout
      - procento vyhovujících řádek v tabulce
      - jejich datový objem



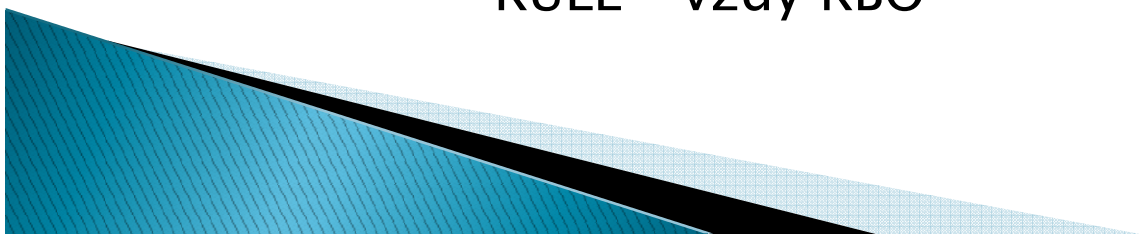
Cost-based optimalizace

- V Oracle dovoluje používat indexy přes výrazy
  - CREATE INDEX Emp_Income_INX
    ON Emp(Sal+COALESCE(Comm,0));
  - SELECT EName FROM Emp
    WHERE Sal+COALESCE(Comm,0)>5000;



Výběr typu optimalizace

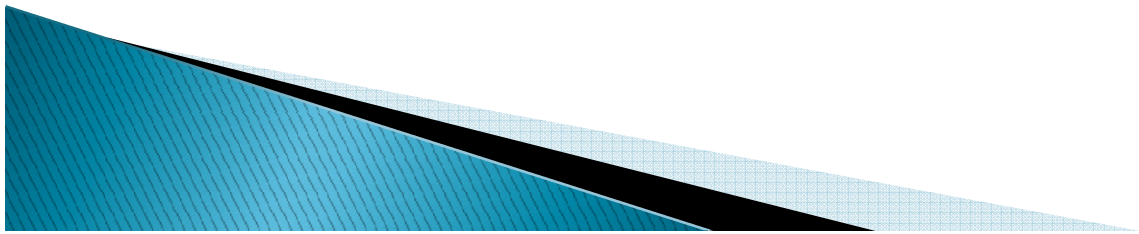
- Výběr optimalizace
  - ALTER SESSION SET OPTIMIZER_GOAL=
    - CHOOSE – výběr podle (ne)přítomnosti statistik nejsou-li k dispozici, potom RBO, jinak CBO
    - ALL_ROWS – vždy CBO, minimalizuje se cena za získání všech řádek odpovědi
      - Vhodné pro dávkové zpracování.
    - FIRST_ROWS – vždy CBO, minimalizuje se cena za získání prvních řádek odpovědi.
      - Vhodné pro interaktivní zpracování.
    - RULE – vždy RBO



Tvorba statistik

- ANALYZE TABLE jm_tabulky {COMPUTE | ESTIMATE | DELETE} STATISTICS [FOR {TABLE | ALL [INDEXED] COLUMNS}];
- DBMS_UTILITY.ANALYZE_SCHEMA('schema');
- DBMS_STATS.GATHER_SCHEMA_STATS;

- Pohledy v datovém slovníku
  - INDEX_STATS,
    USER_TAB_COL_STATISTICS
    USER_USTATS



Nápověda optimalizátoru (hinty)

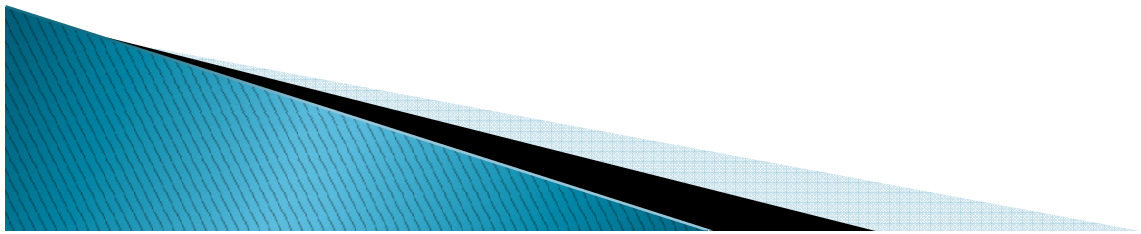
- Pomocí plusových komentářů bezprostředně za prvním klíčovým slovem příkazu SELECT/UPDATE/INSERT/DELETE
  - SELECT --+ seznam hintů
  - SELECT /*+ seznam hintů */
  - Možné použít pro volbu typu optimalizace
    - SELECT /*+ RULE */ * FROM EMP …;
    - SELECT /*+ FIRST_ROWS */ * FROM EMP …;
- Použití hintu (kromě RULE) vždy použije CBO na základě statistik. Pokud nejsou statistiky spočteny, výsledek optimalizace je kontraproduktivní.

Nápověda optimalizátoru (hinty)

- CHOOSE, RULE,
  ALL_ROWS, FIRST_ROWS, FIRST_ROWS(*n*)
- Další možnosti (výběr)
  - FULL(jm_tabulky)
    - Full-scan pro tabulku
  - INDEX (jm_tabulky jm_indexu)
    - Pro průchod tabulkou se použije požadovaný index
  - NO_INDEX (jm_tabulky jm_indexu)
    - Pro průchod tabulkou se nepoužije požadovaný index
  - CLUSTER (jm_clusteru)
    - Pro spojení tabulek se použije cluster
  - ORDERED
    - Při spojování tabulek se použije pořadí uvedené ve FROM
  - USE_NL, USE_MERGE, USE_HASH
    - Spojení pomocí vnořených cyklů, merge-join, hash-join

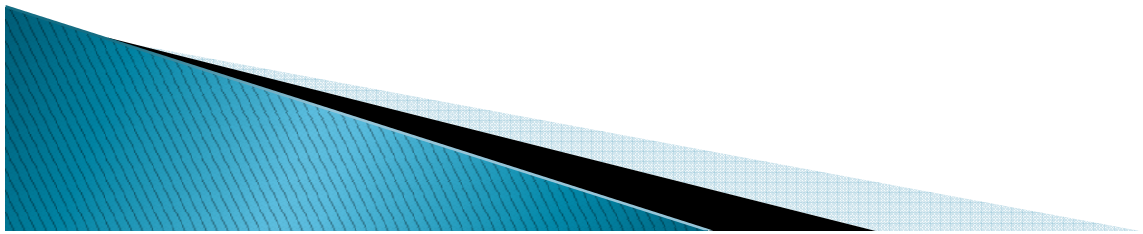
Nápověda optimalizátoru (hinty)

- FULL(jm_tabulky)
  - SELECT /*+ FULL(Emp) */
    EmpNo, Ename
    FROM Emp
    WHERE EName>'X';
  - Použít FULL SCAN
    i přes případně malé procento vracených řádek



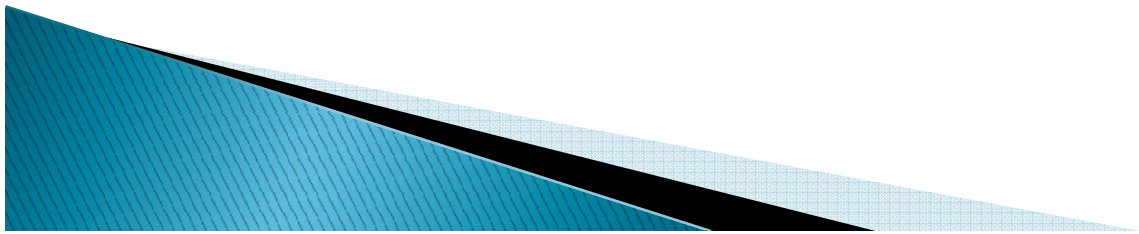
Nápověda optimalizátoru (hinty)

- INDEX(jm_tabulky index [index …])
  - SELECT --+ INDEX(Emp ENameInx EDeptInx)
    EmpNo, Ename
    FROM Emp
    WHERE EName LIKE'SC%' AND DeptNo>50;
  - Použít ten z uvedených indexů, který nabízí nejnižší cenu provedení operace



Nápověda optimalizátoru (hinty)

- NO_INDEX(jm_tabulky index [index …])
  - SELECT --+ NO_INDEX(Emp ENameInx)
    EmpNo, Ename
    FROM Emp
    WHERE EName LIKE'SC%' AND DeptNo>50;
  - Index přes EName se nebude pro vytvoření plánu uvažovat



Nápověda optimalizátoru (hinty)

- ORDERED
  - SELECT --+ ORDERED
    EmpNo, Ename
    FROM Emp, Dept
    WHERE …;
  - Tabulky se budou spojovat v pořadí, v jakém jsou uvedené ve FROM klauzuli