PA152: Efektivní využívání DB

12. ID, indexy a zabezpečení DB

Vlastislav Dohnal

Poděkování

- Zdrojem materiálů tohoto předmětu jsou:
  - Přednášky CS245, CS345, CS345
    - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
    - Stanford University, California
  - Přednáška CS145 podle knihy
    - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom: Database Systems: The Complete Book
  - Kniha
    - Andrew J. Brust, Stephen Forte: Mistrovství v programování SQL Serveru 2005
  - Materiály knihovny MSDN firmy Microsoft

Osnova

- Generování ID
- Prostorová data
  - Datové typy, indexy
- Zabezpečení DB
  - Přístupová práva v DB
  - Uložené procedury
  - Útoky na DB

Generování primárního klíče

- Obvykle rostoucí posloupnost čísel
- Příklad:
  - student(učo, jméno, příjmení)
- Ad-hoc přístup 1:
  - Zjištění aktuálního maxima
    maxučo := SELECT max(učo) FROM student;
  - Zvýšení o jedna a uložení nového záznamu
    INSERT INTO student
    VALUES (maxučo+1, ‘Pepík’, ‘Všeuměl’);
  - Nevýhoda:
    - Souběžný přístup → duplicitní hodnota

Generování primárního klíče

- Ad-hoc přístup 2:
  - Spojení INSERT a SELECT dohromady
    INSERT INTO student VALUES (
    (SELECT max(učo) FROM student)+1,
    ‘Pepík’, ‘Všeuměl’);
  - Aktualizace indexu je atomická
    - Zdánlivě ok….
    - Vnořený select vyhodnocený na “starých datech”
  - Problém duplicitních hodnot méně pravděpodobný.
    - Zlepšení pouze v rychlosti
      - Tj. pouze „jeden“ příkaz

Generování primárního klíče

- Přístup 2: problémy při paralel. zpracování
  - Vždy při spouštění v transakci
  - Záleží na způsobu zamykání v DB:
    - SELECT zamkne data (sdílený zámek)
      - Ostatní jsou blokováni
      - Zámky uvolněny až po commit
    - Příkaz INSERT provede vložení
    - → hodnoty jsou správné, ale ostatní čekají

Generování primárního klíče

- Ad-hoc přístup 3:
  - Vytvoření pomocné tabulky klíče(tabulka VARCHAR, id INTEGER)
    1. UPDATE klíče SET id=id+1 WHERE tabulka='student';
    2. newid := SELECT id FROM klíče WHERE tabulka='student';
       - Popř. společně oba kroky:
         newid := UPDATE klíče SET id=id+1 WHERE tabulka='student' RETURNING id;
    3. INSERT INTO student VALUES (newid , 'Pepík', 'Všeuměl');

Generování primárního klíče

- Ad přístup 3:
  - Nevýhoda při paralelním zpracování v transakci:
    - Příkaz UPDATE zamkne řádek tabulky *klíče*
    - Zámek je uvolněn až po commit (po INSERT)
    - → hodnoty jsou správné, ale ostatní jsou blokováni
  - Výhoda:
    - při aplikaci principu z přístupu 1
      - tj. samostatné transakce
    - → hodnoty jsou správné a ostatní *nejsou* blokováni

Generování primárního klíče

- Nástroje DB – doporučeno využívat
  - Datový typ
    - PostgreSQL: SERIAL, BIGSERIAL
    - SQLServer: IDENTITY
  - Sekvence
    - Oracle, PostgreSQL
  - Přepínač atributu
    - MySQL
- Umožňují i zjištění vygenerovaného čísla
  - Lze jej použít pro ukládání do více tabulek
    - Např. vložení prvního zboží do košíku v e-shopu
      - tj. vytvoření košíku & vložení zboží

Generování primárního klíče

- Sekvence (CREATE SEQUENCE …)
  - Generátor posloupnosti čísel
  - Lze různě nastavit
    - min. a max. hodnota, cyklická
- Funkce v PostgreSQL
  - Nextval – nová hodnota sekvence
  - Currval – posledně vrácená hodnota sekvence
  - Lze použít přímo v INSERT
    - INSERT INTO tabulka
      VALUES (nextval('sekvence'), …);

Generování PK: výkonnost

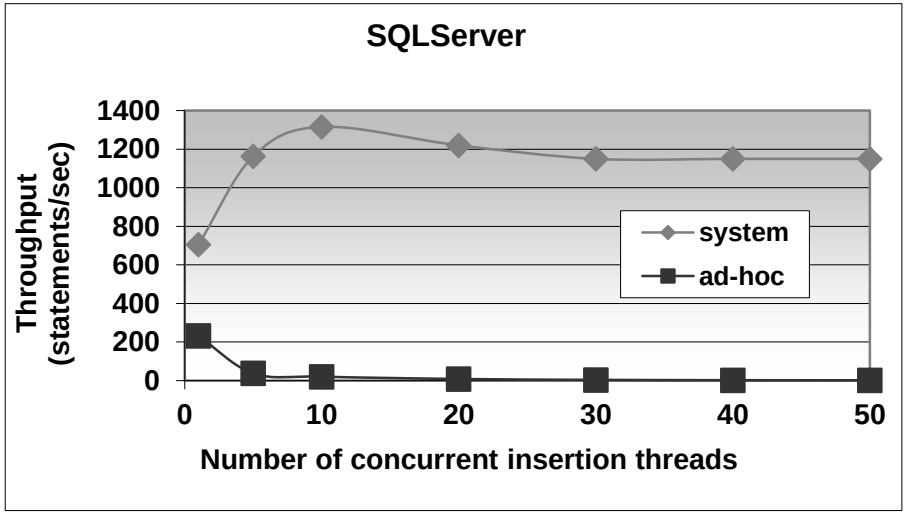
- Příklad pro ad-hoc přístup 3:
  - accounts(number, branchnum, balance);
    - Shlukovaný index nad *number*
  - counter(nextkey);
    - Vložen jeden záznam s hodnotou 1
    - Pro přidělování hodnoty *number* pomocí ad-hoc metody 3
- Konfigurace:
  - Nastavení transakce: READ COMMITTED
    - Viditelné pouze změny potvrzené commitem.
  - Dual Xeon (550MHz,512Kb), 1GB RAM, RAID controller, 4x 18GB drives (10000RPM), Windows 2000.

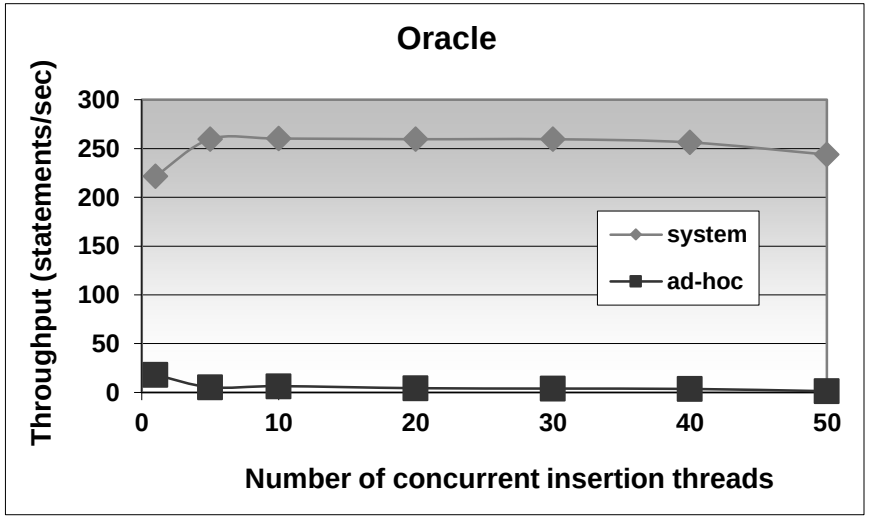
Generování PK: výkonnost

- Dávka: vlož 100 000 záznamů do *accounts*
- Generování ID:
  - Podpora DB:
    - SQLServer 7 (identity)
      - insert into accounts (branchnum, balance) values (94496, 2789);
    - Oracle 8i (sekvence)
      - insert into accounts values (seq.nextval, 94496, 2789);
  - Ad-hoc přístup 3:

```
begin transaction
    update counter  set nextkey = nextKey+1;
    :nk := select nextkey  from counter;
commit transaction
begin transaction
    insert into accounts values (:nk, 94496, 2789);
commit transaction
```

Generování primárního klíče





- Osa X:
  - Zvyšující se počet paralelních vkládání
- *system* (podpora DB) vítězí nad *ad-hoc.*

Generování primárního klíče

- PostgreSQL
  - CREATE TABLE vyrobek (
    id SERIAL PRIMARY KEY,
    nazev VARCHAR(10)
    );
  - Vnitřní implementace
    - Vytvořena sekvence
      - vyrobek_id_seq
    - Nastavena implicitní hodnota atributu *id*
      - nextval('vyrobek_id_seq')

Generování primárního klíče

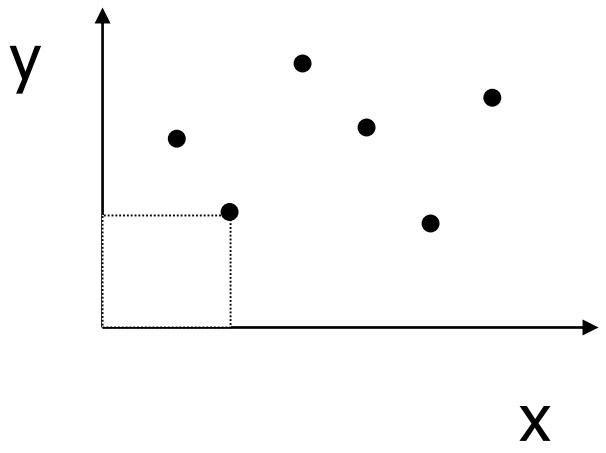
- PostgreSQL (ručně)
  - CREATE SEQUENCE vyrobek_id_seq;
  - CREATE TABLE vyrobek (
    id INT PRIMARY KEY
    DEFAULT nextval('vyrobek_id_seq'),
    nazev VARCHAR(10)
    );
- Používání:
  - INSERT INTO vyrobek (nazev)
    VALUES ('Cívka');
  - INSERT INTO vyrobek (id, nazev)
    VALUES (DEFAULT, 'Cívka');

Osnova

- Generování ID
- **Prostorová data**
  - **Datové typy, indexy**
- Zabezpečení DB
  - Přístupová práva v DB
  - Uložené procedury
  - Útoky na DB

Zpracování prostorových dat

- Prostorová data
  - Obvykle geografická, 2d geometrická
    - Souřadnice X, Y



<X_1,Y_1, Attributes>
<X_2,Y_2, Attributes>
...

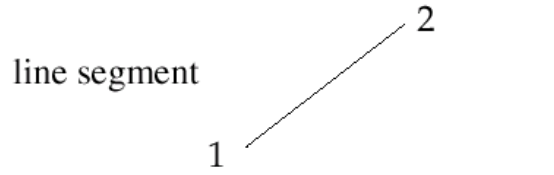
Zpracování prostorových dat

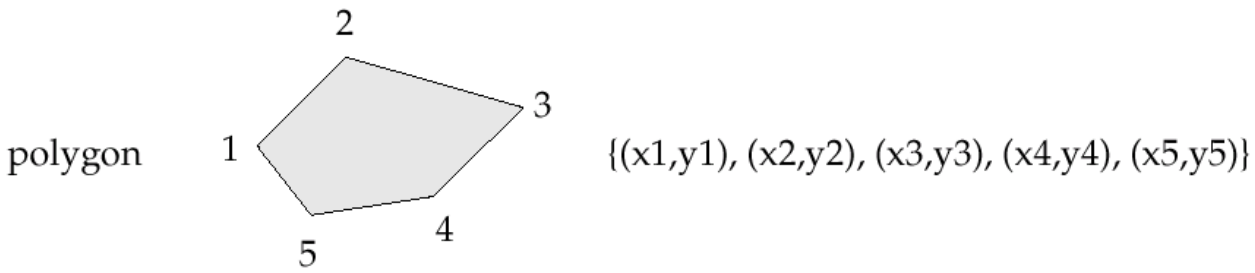
- Typické dotazy
  - Jaké je město na pozici $<X_i, Y_i>$?
  - Co se vyskytuje v okolí 5 km od $<X_i, Y_i>$?
  - Jaké je nejbližší místo (uložené v DB) k bodu $<X_i, Y_i>$?

- Bez podpory DB
  - Jak měřit vzdálenost? (např. u GPS souřadnic)
    - Naimplementovat jako uživatelskou funkci
  - Index na X, popř. na XY apod.
    - Problematické

Zpracování prostorových dat

- Jiná data:
  - úsečky, obdélníky, regiony, …





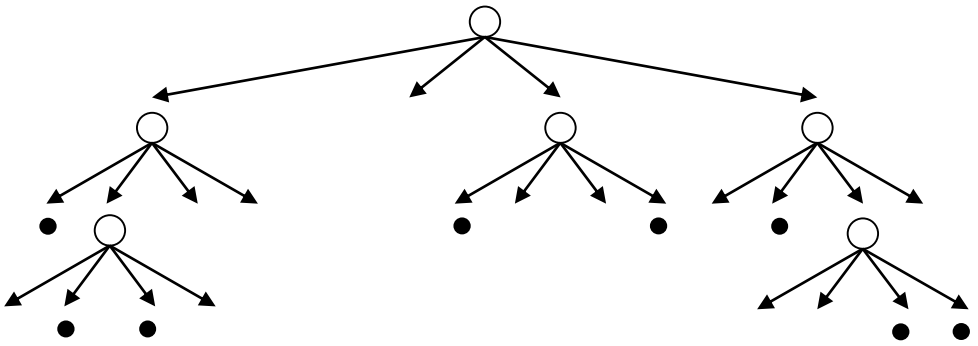
- Dotazy:
  - Je bod součástí regionu? Protínají se regiony? …

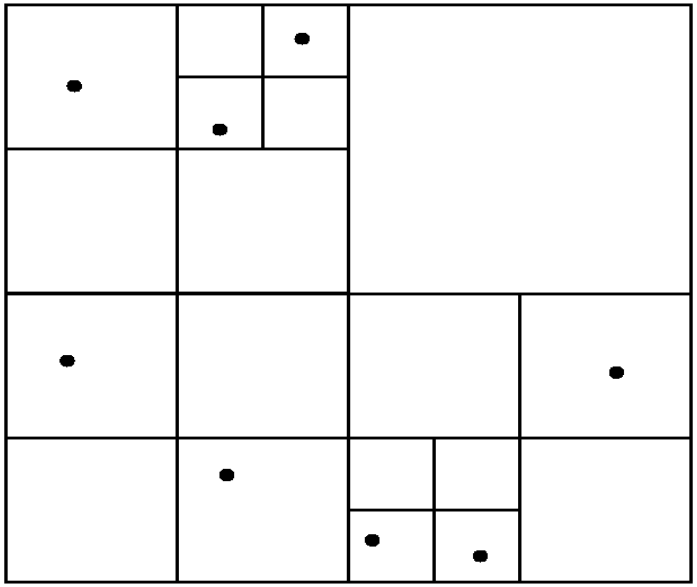
Zpracování prostorových dat

- Vhodná je podpora DB
  - Speciální datové typy a funkce
    - PostgreSQL
      - Typy: point, line, box, circle, …
      - Funkce: area(), center(), length(), …
      - Operátory: **~=** *same as*, **~** *contains*, **?#** *intersects*, …
      - Index: R-strom
    - SQL Server 2008
      - Typy: point, linestring, polygon, geography, …
      - Index: Grid s využitím B-stromů
    - Oracle 9i
      - Typy: SDO_GEOMETRY (SDO_POINT, SDO_LINE,…)
      - Index: R-strom, Quad-strom

Zpracování prostorových dat

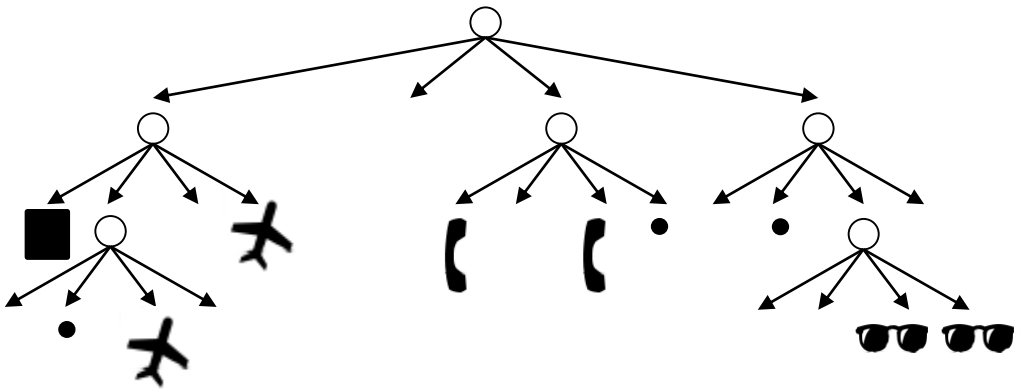
- Quad-strom
  - Vyhledávací strom, každý uzel dělí prostor do 2^d stejných oblastí (např. 2d data $\rightarrow$ 4 oblasti)
  - Listové uzly mohou mít větší kapacitu

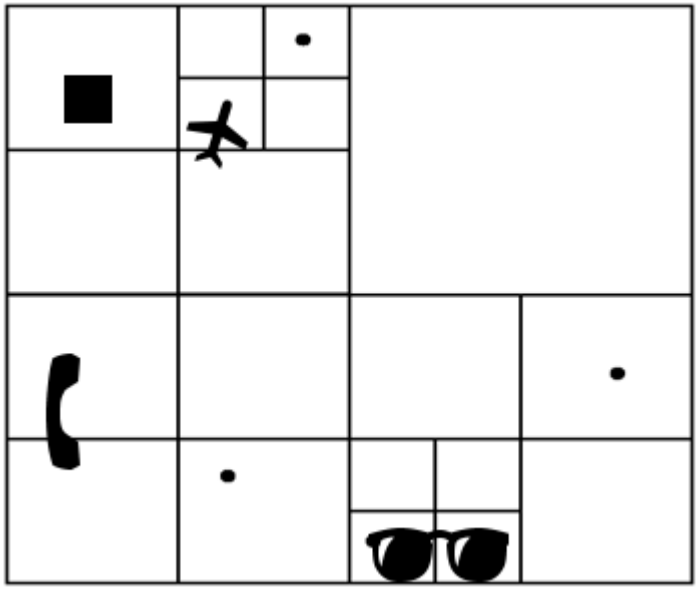




Zpracování prostorových dat

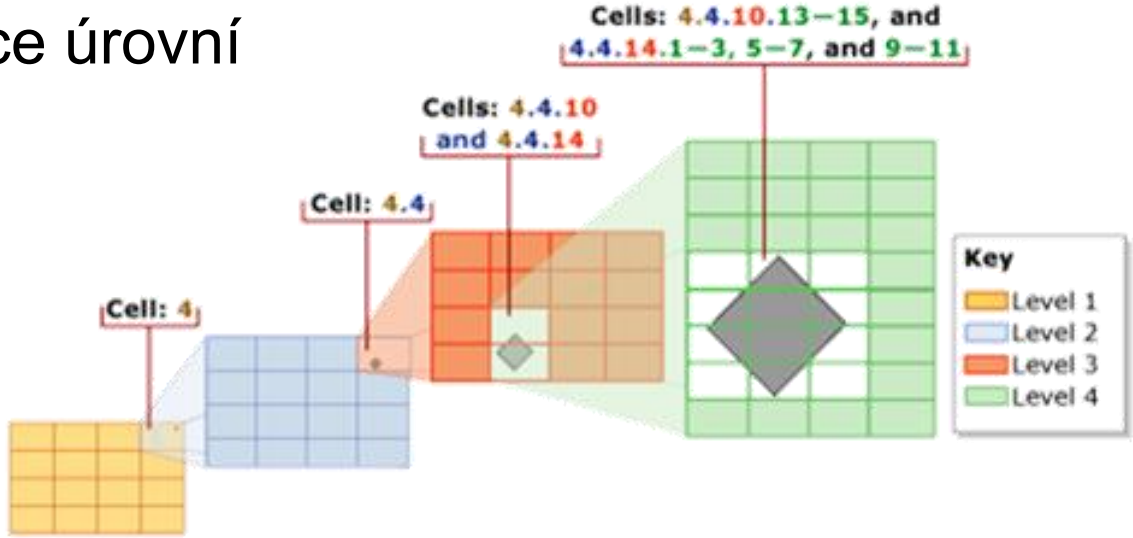
- Quad-strom
  - Pouze pro body
  - Rozšíření na regiony:
    - Region je vložen do více oblastí
    - Složité objekty obaleny obdélníkem





Zpracování prostorových dat

- Grid (mřížka)
  - Prostor ohraničen: x_{min}, y_{min}, x_{max}, y_{max}
  - SQL Server
    - Rozdělení na pevný počet buněk 4x4, 8x8, 16x16
    - Více úrovní



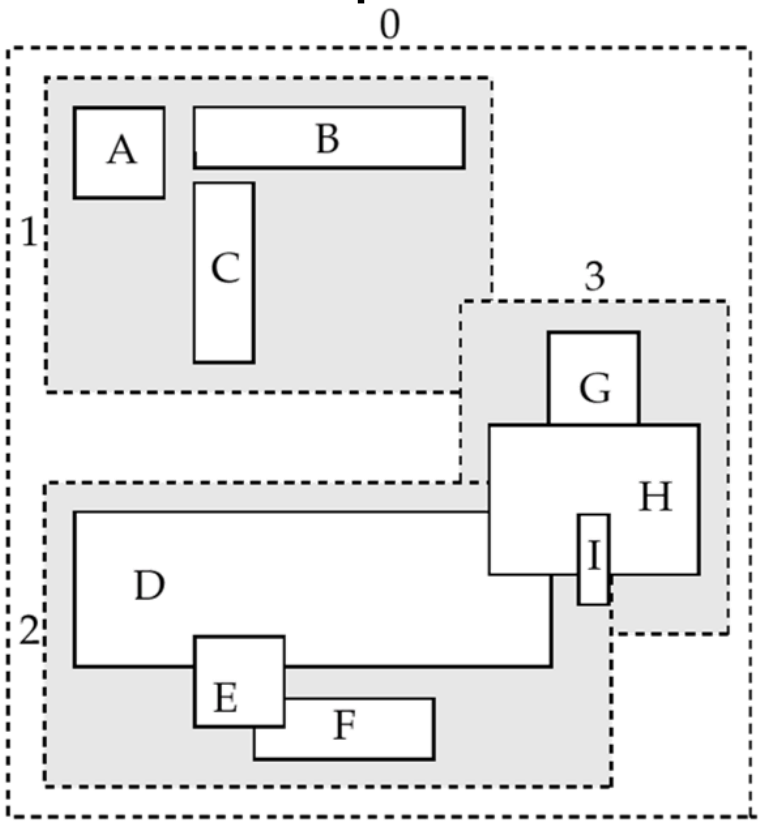
Zdroj: Microsoft MSDN, http://msdn.microsoft.com/en-us/library/bb964712.aspx

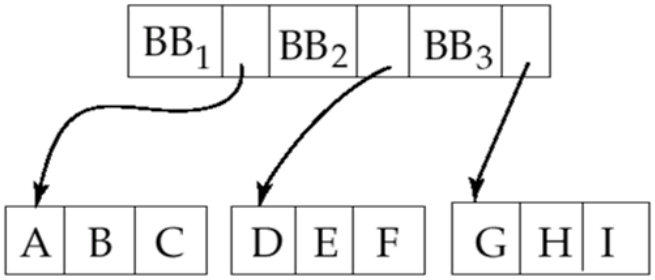
Zpracování prostorových dat

- R-strom (Rectangle Tree)
  - Rozšíření B^+ stromů na *d* rozměrná data
    - Vkládání, mazání je v podstatě stejné
  - List obsahuje několik datových prvků
    - List je popsán minimálním pokrývajícím obdélníkem (*minimum bounding rectangle - MBR*)
  - Vnitřní uzly
    - odkazují na potomky a jejich MBR
  - MBR uzlů se mohou překrývat → vyhledávání pak musí následovat všechny odpovídající větve stromu.
  - Každý datový prvek je uložen pouze jednou
    - Výhoda oproti Grid, Quad-stromu

Zpracování prostorových dat

- R-strom
  - ukládání jiných dat pomocí MBR (objekt jím obalím a pak uložím do stromu)





Osnova

- Generování ID
- Prostorová data
  - Datové typy, indexy
- **Zabezpečení DB**
  - **Přístupová práva v DB**
  - Uložené procedury
  - Útoky na DB

Přístupová práva

- Analogie se souborovým systémem
  - Objekty
    - Soubor, adresář, …
  - Subjekty
    - Typicky: vlastník, skupina, ostatní
  - Přístupová práva
    - Definována na objektu *O* pro subjekt *S*
    - Typicky: čtení, zápis, spouštění

Přístupová práva

- Databáze
  - Obvykle jemnější práva než u souborového systému
  - Specifická práva pro
    - Tabulky, pohledy, sekvence, schéma, databáze, procedury, …
  - Pohledy (views)
    - základní nástroj pro řízení přístupu
  - Subjektem jsou obvykle uživatelé a skupiny
    - Často nazýváno jako *authorization id* nebo *role*
    - Subjekt „ostatní“ je označován jako PUBLIC
      - Povolení přístupu pro PUBLIC znamená povolení přístupu komukoli.

Přístupová práva

- Práva pro relace (tabulky)
  - SELECT – čtení obsahu (tj. výběr řádků)
    - Někdy lze omezit na vybrané atributy
  - INSERT – vkládání řádků
    - Někdy lze omezit na vybrané atributy
  - DELETE – mazání řádků
  - UPDATE – aktualizace řádků
    - Někdy lze omezit na vybrané atributy
  - REFERENCES – vytvoření cizího klíče

Přístupová práva

- Příklad
  - INSERT INTO Beers(name)

```
SELECT beer FROM Sells
WHERE NOT EXISTS
        (SELECT * FROM Beers
          WHERE name = beer);
```

Vložení názvů piv, které ještě nemám v evidenci.

  - Požadavky:
    - INSERT pro relaci *Beers*
    - SELECT pro relace *Sells* a *Beers*

Přístupová práva

- Omezení přístupu pomocí pohledu
  - Relace
    - Zamestnanci(id, jmeno, adresa, plat)
  - Chceme skrýt výši platu:
    - CREATE VIEW ZamestnanciAdresa AS
      SELECT id, jmeno, adresa
      FROM Zamestnanci;
    - Práva:
      - Odebrání práva SELECT na relaci Zamestnanci
      - Přidání práva SELECT na ZamestnanciAdresa

Přístupová práva

- Udílení práv
  - GRANT <list of privileges>
    ON <relation or object>
    TO <list of authorization ID's>;
- Lze povolit i „udílení práv“ oprávnění
  - Připojí se fráze „WITH GRANT OPTION“
    - GRANT SELECT
      ON TABLE ZamestnanciAdresa
      TO karel
      WITH GRANT OPTION

Přístupová práva

- Příklad
  - Jako vlastník relace *Sells* provedu
    - GRANT SELECT, UPDATE(price)
      ON sells TO sally;
- Uživatel *sally* může
  - zobrazovat obsah relace *Sells*
  - měnit obsah atributu *price*

Přístupová práva

- Příklad
  - Jako vlastník relace *Sells* provedu
    - GRANT UPDATE ON sells TO sally WITH GRANT OPTION;
- Uživatel *sally* může
  - měnit libovolný atribut relace *Sells*
  - udělovat oprávnění dalším uživatelům
    - Lze udělit pouze UPDATE oprávnění, např. omezené na jednotlivé atributy.

Přístupová práva

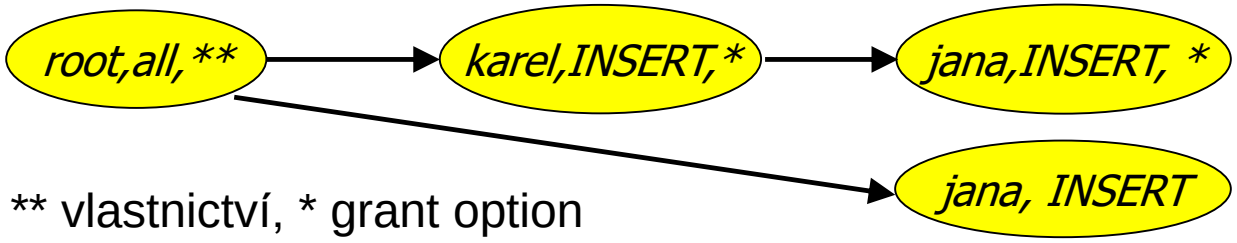
- Odebírání práv
  - REVOKE <list of privileges>
    ON <relation or object>
    FROM <list of authorization ID's>;
- Daným uživatelům je odebráno určité oprávnění.
  - Pozor, uživatelé ale stále mohou mít přístup povolený
  - → protože jim byl udělený ještě někým jiným.
    - nebo plyne z povolení pro PUBLIC / členství v nějaké skupině

Přístupová práva

- Odebírání práv
  - Přidání za REVOKE
    - CASCADE – zruší i oprávnění povolené uživatelem, kterému právě oprávnění odebírám
    - RESTRICT (implicitní volba) – odebere pouze toto oprávnění
      - Pokud existují další oprávnění udělená uživatelem, kterému právo odebírám, příkaz skončí chybou.
  - REVOKE GRANT OPTION FOR …
    - Zruší pouze povolení udělovat oprávnění dále.
    - Bez tohoto modifikátoru je odebráno oboje.

Přístupová práva – diagram

- Diagramy reprezentují práva udělená kým a komu



  - Každý objekt má vlastní diagram
  - Uzel je určen
    - Rolí (uživatelem / skupinou)
    - Uděleným právem
    - Povolením udělovat právo dál / vlastnictvím objektu
  - Hrana mezi X a Y
    - X bylo použito pro udělení oprávnění Y

Přístupová práva – diagram

- „*root,all* “ označuje
  - uživatel *root* má oprávnění *all*.
- Oprávnění „*all*“ pro tabulku
  - = insert, update, delete, select, references
- Oprávnění „*“ (*grant option*)
  - oprávnění s povolením udílení oprávnění dalším
- Oprávnění „**“
  - zdroj vzniku oprávnění (vlastník objektu)
- Vlastník objektu
  - dovoleno vše
  - toto implikuje povolení udílet oprávnění dalším

Přístupová práva – diagram

- Vytváření hran
  - Když *A* udílí *P* dalšímu uživateli *B*, pak vytváříme hranu z *AP** nebo z *AP*** do *BP*.
    - Popř. do *BP**, pokud je „*with grant option*“.
  - Když *A* uděluje nižší oprávnění *Q* než je *P*, pak hrana vede do uzlu *BQ* (popř. *BQ**).

Přístupová práva – diagram

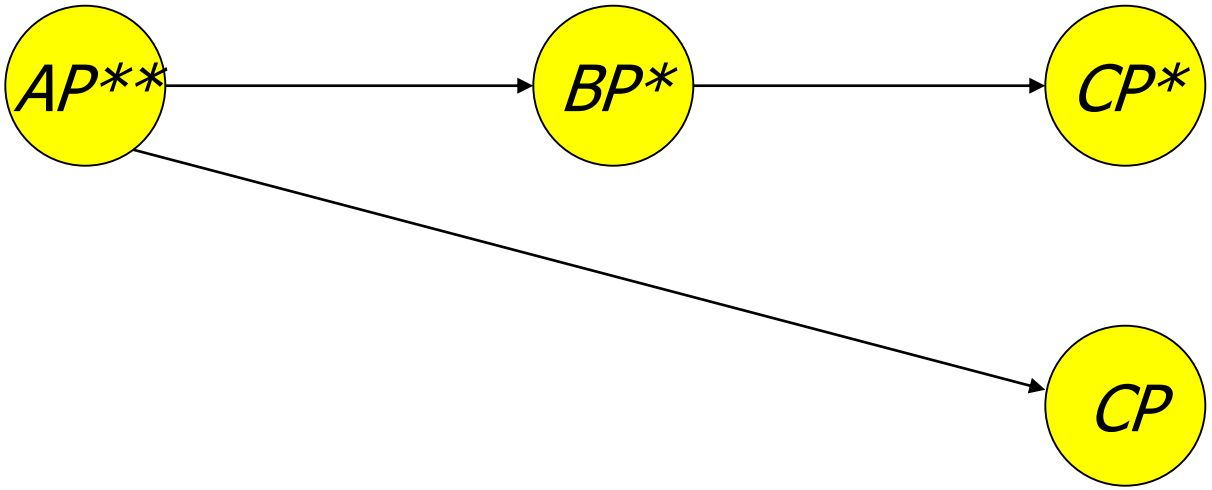
- Test oprávnění
  - Uživatel *C* má oprávnění *Q*, pokud v grafu existuje uzel *OP*, *OP** nebo *OP***
    - kde *P* je vyšší oprávnění než *Q* nebo stejné
    - kde *O* = *C* nebo *C* je členem skupiny *O*

Přístupová práva – diagram

A vlastní objekt s oprávněním *P*.

A:
GRANT P TO B
WITH GRANT OPTION

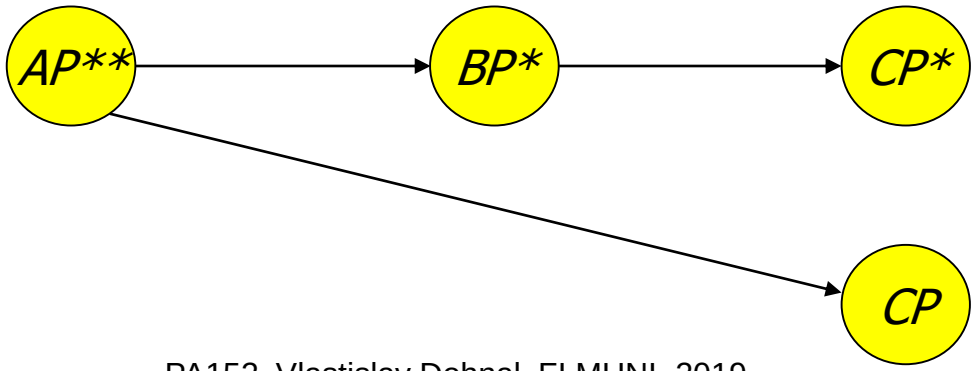
B:
GRANT P TO C
WITH GRANT OPTION



A:
GRANT P TO C

Přístupová práva – diagram

- Odebrání oprávnění
  - A ruší oprávnění P pro subjekt B
    - Test, zda existuje hrana AP $\rightarrow$ BP.
    - Pokud ano, hrana se zruší.
  - Pokud B povolilo P dalšímu subjektu, je nutné použít CASCADE.



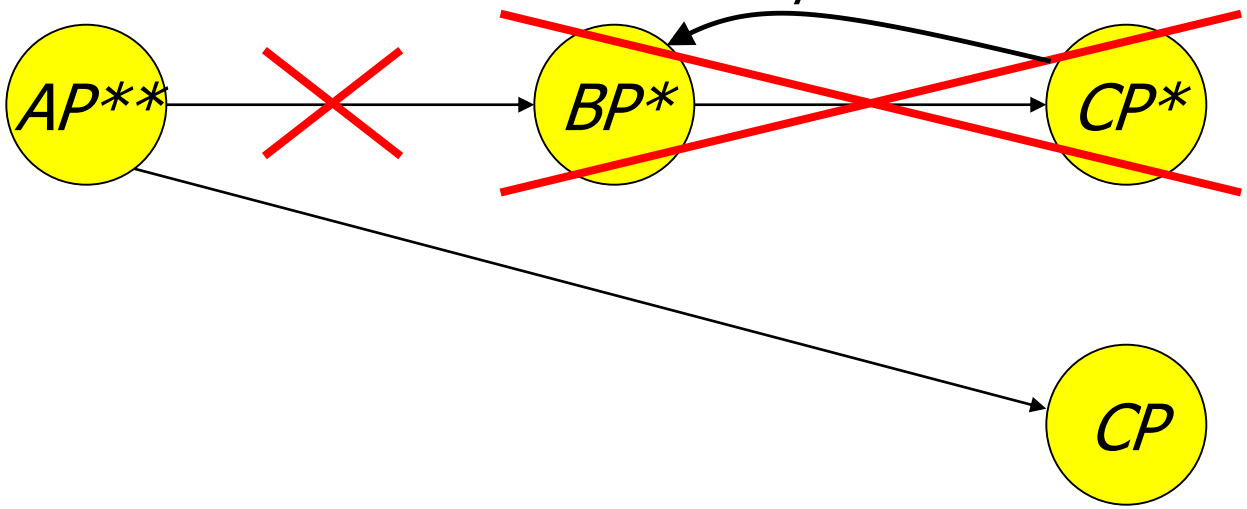
Přístupová práva – diagram

- Odebrání oprávnění
  - Po smazání hrany se musí otestovat
    - zda neexistují uzly, které nejsou dosažitelné z nějakého ** uzlu (tj. od vlastníka).
  - Pokud nějaký takový uzel existuje, je z diagramu smazán
    - včetně hran z něj vycházejících

Přístupová práva – diagram

A:
REVOKE P FROM B
CASCADE

Nejenom, že *B* ztratí *P**, ale i *C* ztratí *P**. Uzly *BP** a *CP** jsou smazány.



Uzly jsou smazány, i když *C* udělilo *P** zpět *B*.

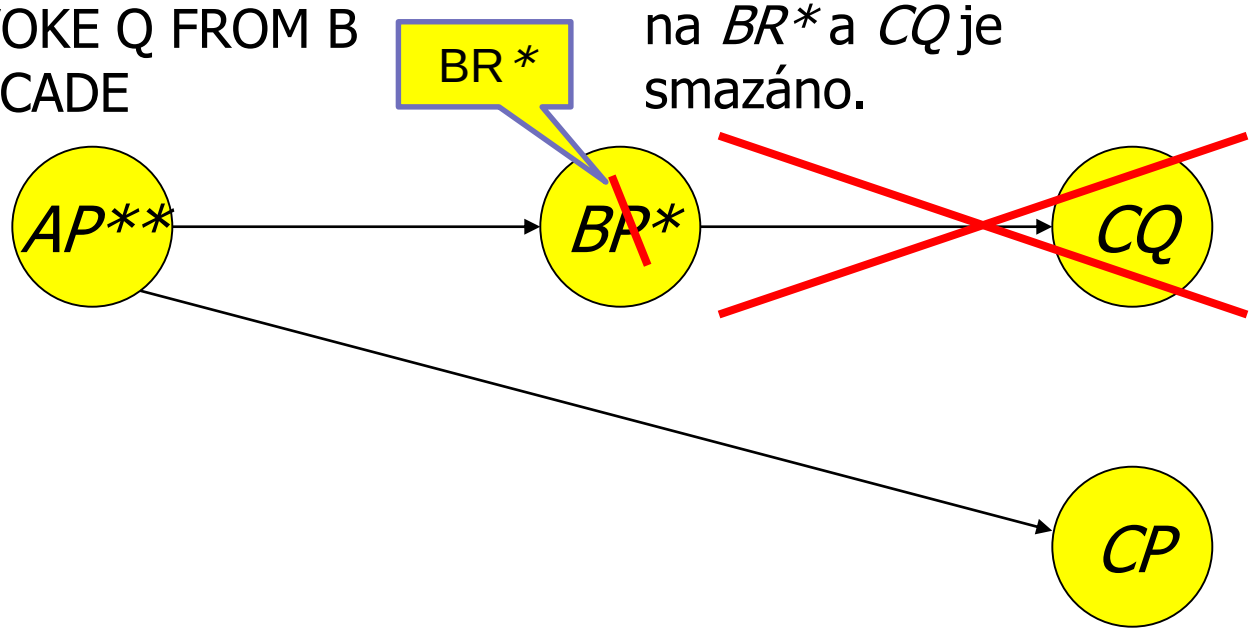
Avšak *C* bude stále mít oprávnění *P*, protože jej získalo také přímo od *A*.

Přístupová práva – diagram

P = {Q*,R*}

A:
REVOKE Q FROM B
CASCADE

Nejenom, že *B* ztratí *Q*, ale *BP** se změní na *BR** a *CQ* je smazáno.



Osnova

- Generování ID
- Prostorová data
  - Datové typy, indexy
- Zabezpečení DB
  - Přístupová práva v DB
  - **Uložené procedury**
  - Útoky na DB

Uložené procedury

- Vlastní kód provádějící nějakou činnost
  - Např. výpočet faktoriálu, vzdálenost GPS souřadnic, vypočítání průměrného platu, vkládání řádků do různých tabulek, …
- PostgreSQL
  - CREATE FUNCTION název ([parametry,…])
    [RETURNS typ]
    kód funkce

Uložené procedury

- Příklad:
  - Výpočet průměrného platu bez zveřejnění jednotlivých platů
    - Relace Zamestnanci(id, jmeno, adresa, plat)
  - PostgreSQL:
    - CREATE FUNCTION avgsal() RETURNS real AS ‘SELECT avg(plat) FROM zamestnanci’ LANGUAGE SQL;
  - Uživatel použije pro získání průměru:
    - SELECT avgsal();

Uložené procedury

- Příklad (pokr.):
  - Takové řešení nám ale platy *nezabezpečí*
  - Je nutné provést
    - REVOKE SELECT ON Zamestnanci FROM …
    - GRANT EXECUTE ON FUNCTION avgsal() TO …
  - Při provádění SELECT avgsal(); je funkce spuštěna pod aktuálním uživatelem
  - → musí mít povolení SELECT pro Zamestnanci

Uložené procedury

- Kontext provádění
  - Nastavení uživatele, kterého oprávnění se použijí
  - Typy:
    - **Volající** – provede se v kontextu uživatele, který proceduru volá (obvykle aktuální uživatel)
    - **Vlastník** – provede se v kontextu vlastníka uložené procedury
    - **„určený uživatel“** – provede se v kontextu daného uživatele
    - …

Uložené procedury

- Kontext provádění
  - PostgreSQL
    - **Volající** – SECURITY INVOKER
    - **Vlastník** – SECURITY DEFINER
- Řešením je nastavit kontext vlastníka
  - CREATE FUNCTION …. LANGUAGE SQL **SECURITY DEFINER**;
    - Předpoklad: vlastník má k relaci Zamestnanci oprávnění SELECT

Útoky na DB systém

- Připojení z internetu
  - Otevřené připojení na DB → používat firewall
- Přihlášení
  - Slabé heslo (zejména správce)
  - Povolení přihlášení uživatele odkudkoli
    - Lze omezit na konkrétní uživatele, IP adresy a databáze
  - Nezabezpečené připojení
    - Šifrovat spojení pomocí SSL (obvykle podporováno)
  - Používání jediného účtu k DB systému

Útoky na DB systém

- SQL injection
  - Útok, kdy uživatel systému zadá příkazy SQL místo platných vstupních údajů ve formuláři aplikace.
  - Souvisí zejména s používáním jediného účtu k DB
    - který má oprávnění správce)-:

SQL injection – příklad

- Aplikace zobrazí formulář pro zadání poznámky:
  - Aplikace připraví příkaz pro databázi:

```
UPDATE zakaznik SET pozn='$poznamka'
WHERE id=current_user;
```

- Uživatel zadá:

```
Vader'; DROP TABLE zakaznik; --
```

- Po doplnění vstupu se provede:

```
UPDATE zakaznik
SET pozn='Vader'; DROP TABLE zakaznik; --'
WHERE id=current_user;
```

Toto je vše na jednom řádku

SQL Injection: Countermeasures

- Používání uživatelských účtů
  - vyloučení používání admin účtu
- Kontrola vstupních hodnot
  - délka vstupu, nevhodné znaky,…
- Funkce progr. jazyka
  - *mysql_real_escape_string()*, *add_slashes()*
  - $dbh->*quote*($string)
- Funkce v DB
  - *quote_literal(str)*
    - returns a string *str* suitably quoted to be used as a string literal in an SQL statement

SQL Injection: Countermeasures

- Prepared statements
  - Parsed statements prepared in DB
    - i.e. compiled templates ready for use
  - Values are then substituted
    - Parameters do not need to be quoted then
  - May be used repetitively
  - Example:

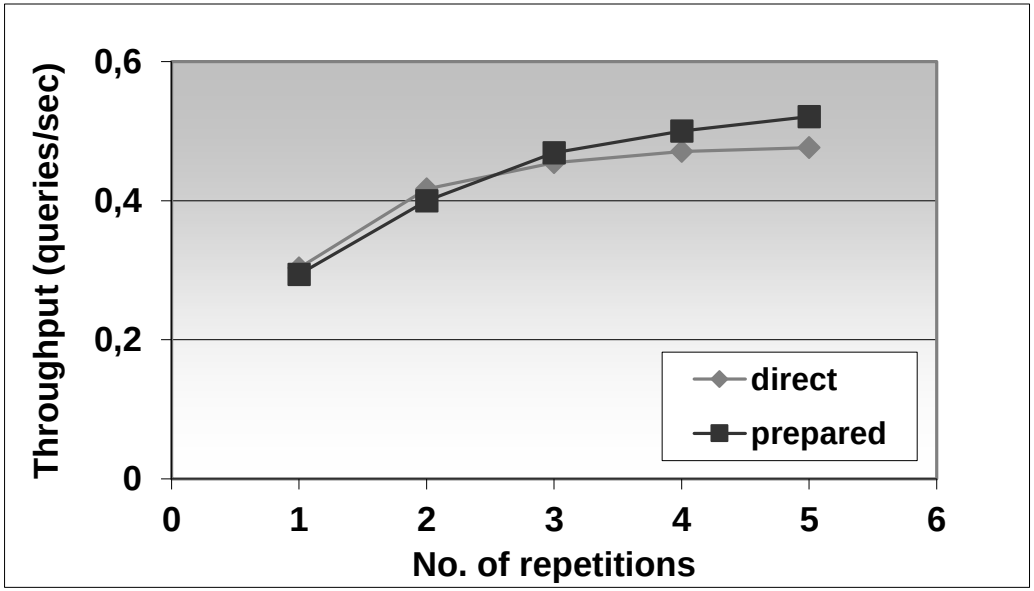
```
$st = $dbh->prepare("SELECT * FROM emp WHERE name LIKE ?");
$st->execute(array( "%$_GET[name]%" ));
```

SQL Injection: Countermeasures

- Prepared statements at server-side
  - The same concept, but stored in DB
  - Typically in procedural languages in DB
  - PostgreSQL
    - PREPARE emp_row(text) AS SELECT * FROM emp WHERE name LIKE **$1**;
      EXECUTE emp_row(**'%John%'**);
- Query is planned in advance
  - Planning time can be amortized
  - But: the plan is generic!
    - i.e. without any optimization induced by knowing the parameter
  - Lasts only for the duration of the current db session

Prepared Statements: Performance

- Prepared execution yields better performance when the query is executed more than once:
  - No compilation
  - No access to catalog.



- Experiment performed on Oracle8iEE on Windows 2000.

Attacking Views

- Views protect data rows…
  - if permissions are correctly set
  - E.g. student(studentid, firstname, lastname, fieldofstudy)
    - CREATE OR REPLACE VIEW studentssme AS SELECT * FROM student WHERE fieldofstudy = 'N-SSME';
  - But, creating a "cheap" function
    - CREATE OR REPLACE FUNCTION test(name text, study text) RETURNS boolean AS $$
      begin
      raise notice 'Name: %, Study: %', name, study;
      return true;
      end;
      $$ LANGUAGE plpgsql VOLATILE COST 0.00001;
  - The query leaks other students in a side channel…
    - SELECT * FROM studentssme WHERE test(lastname, fieldofstudy)
      - NOTICE: Name: Nový, Study: N-AplInf
        NOTICE: Name: Dlouhý, Study: N-Inf
        NOTICE: Name: Svoboda, Study: N-AplInf
        NOTICE: Name: Starý, Study: N-SSME
        NOTICE: Name: Lukáš, Study: N-SSME
        …
- Countermeasure:
  - ban creating new DB objects
  - use security_barrier in Pg.conf or in create view