



PA152: Efektivní využívání DB  
12. Zabezpečení a ...

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', 'Všeználek');`
  - 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', 'Všeználek'  
);
- 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 zámeček)
      - 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)
- Aktualizace poslední hodnoty  
UPDATE klíče SET id=id+1  
WHERE tabulka='student';
- Načtení nového id  
novéid := SELECT id FROM klíče  
WHERE tabulka='student';
- Uložení záznamu  
INSERT INTO student  
VALUES (novéid, 'Pepík', 'Všeználek');

# 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 *id* pomocí ad-hoc metody3

## ■ Konfigurace:

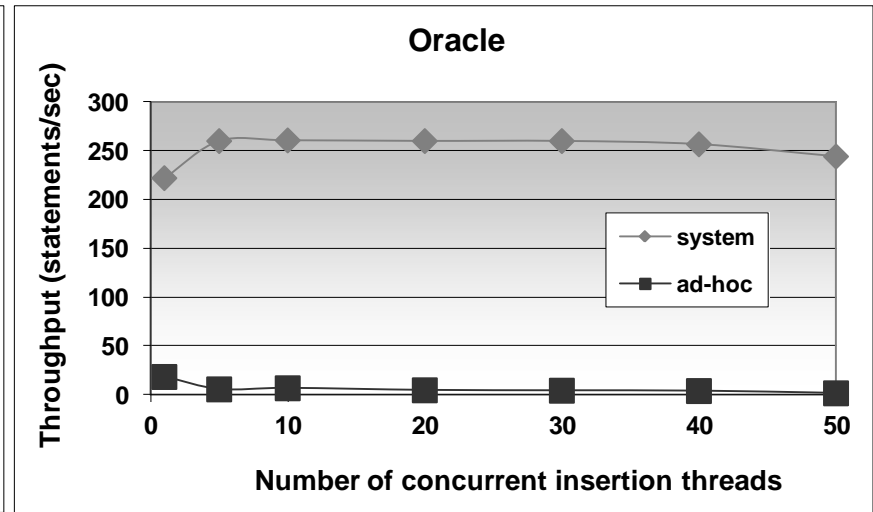
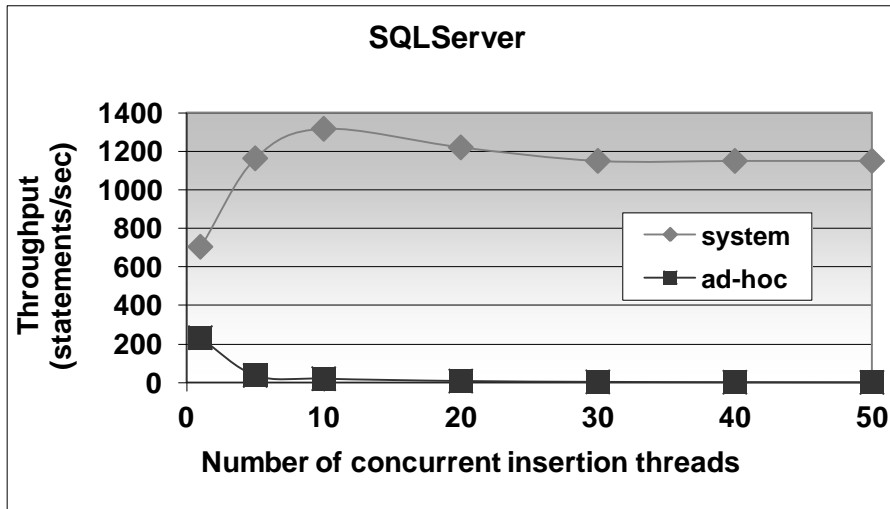
- Nastavení transakce: READ COMMITTED
  - Viditelné pouze změny potvrzené commitem.
- Dual Xeon (550MHz,512Kb), 1GB RAM, RAID controller, 4x 18GB drives (10000RPM), Windows 2000.

# 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

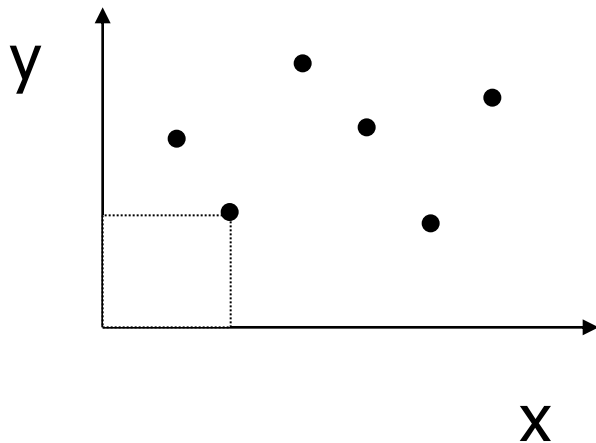


# Zpracování prostorových dat

## ■ Prostorová data

□ Obvykle geografická, 2d geometrická

■ Souřadnice X, Y



$\langle X_1, Y_1, \text{Attributes} \rangle$

$\langle X_2, Y_2, \text{Attributes} \rangle$

...

# Zpracování prostorových dat

## ■ Typické dotazy

- Jaké je město na pozici  $\langle X_i, Y_i \rangle$ ?
- Co se vyskytuje v okolí 5 km od  $\langle X_i, Y_i \rangle$ ?
- Jaké je nejbližší místo (uložené v DB) k bodu  $\langle X_i, Y_i \rangle$ ?

## ■ Bez podpory DB

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

# Zpracování prostorových dat

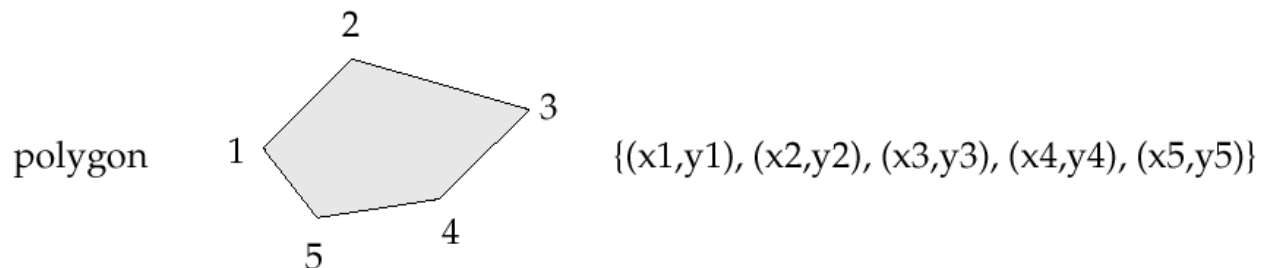
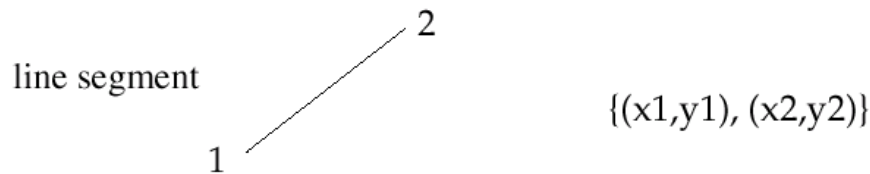
- Jiná data:

- 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:  $\sim =$  same as,  $\sim$  contains,  $? \#$  intersects, ...

- Index: R-strom

- SQL Server 2008

- Index: Grid

- Oracle 9i

- Index: R-strom, Quad-strom

# Zpracování prostorových dat

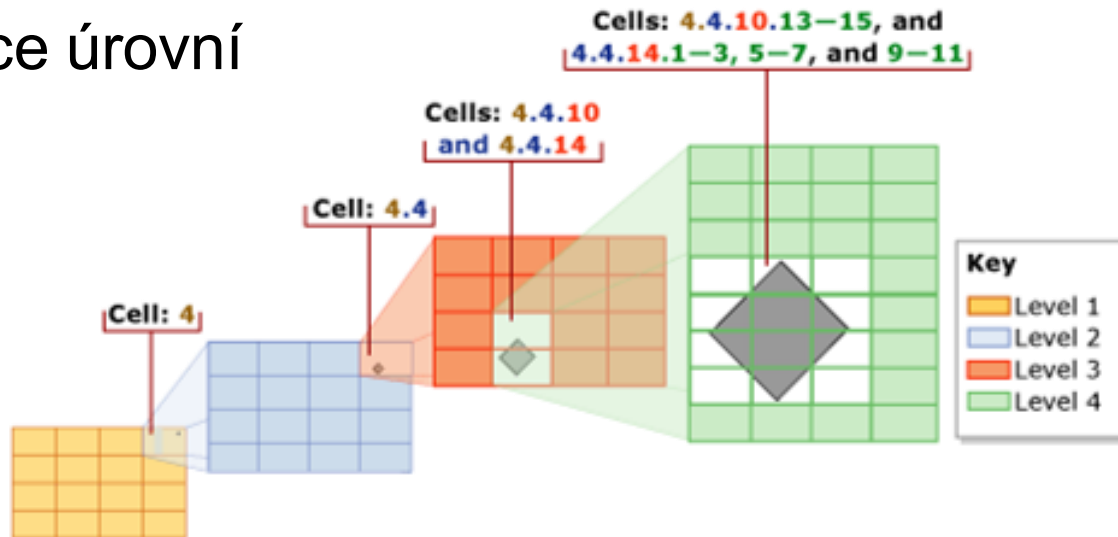
## ■ Grid (mřížka)

□ Prostor ohraničen:  $x_{\min}$ ,  $y_{\min}$ ,  $x_{\max}$ ,  $y_{\max}$

□ SQL Server

■ Rozdělení na pevný počet buněk 4x4, 8x8, 16x16

■ Více úrovní

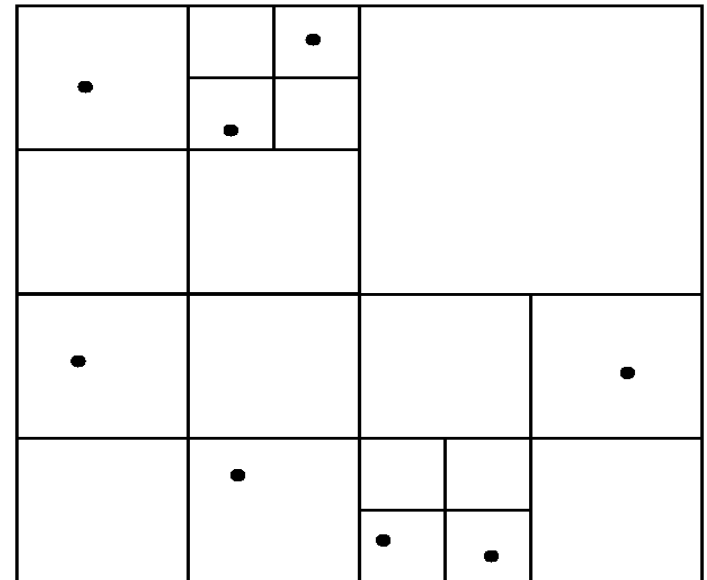
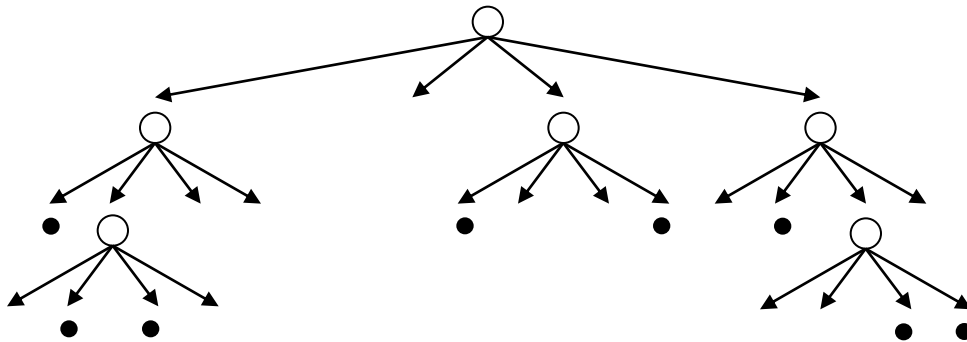


Převzato z Microsoft MSDN: <http://msdn.microsoft.com/en-us/library/bb964712.aspx>

# Zpracování prostorových dat

## ■ Quad-strom

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



# Zpracování prostorových dat

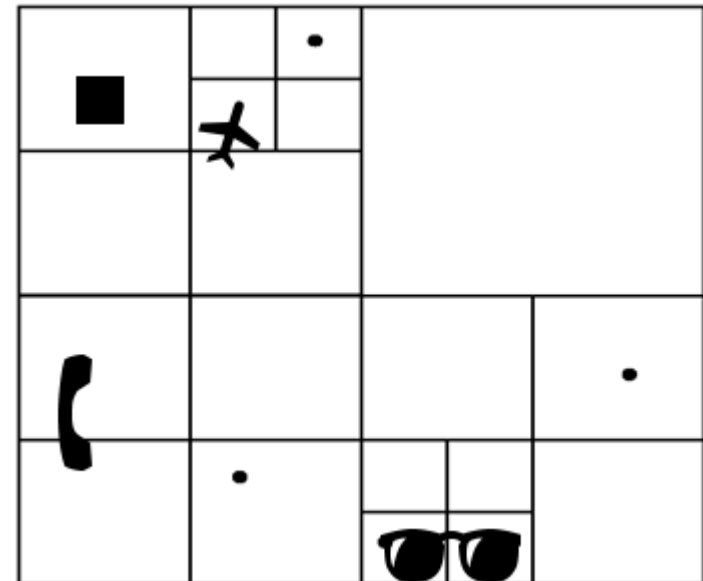
## ■ Quad-strom

□ Pouze pro body

□ Rozšíření na regiony:

■ Region je dělen podle hranic „kvadrantů“

■ Složité objekty  
obaleny obdélníkem



# Zpracování prostorových dat

## ■ R-strom (Rectangle Tree)

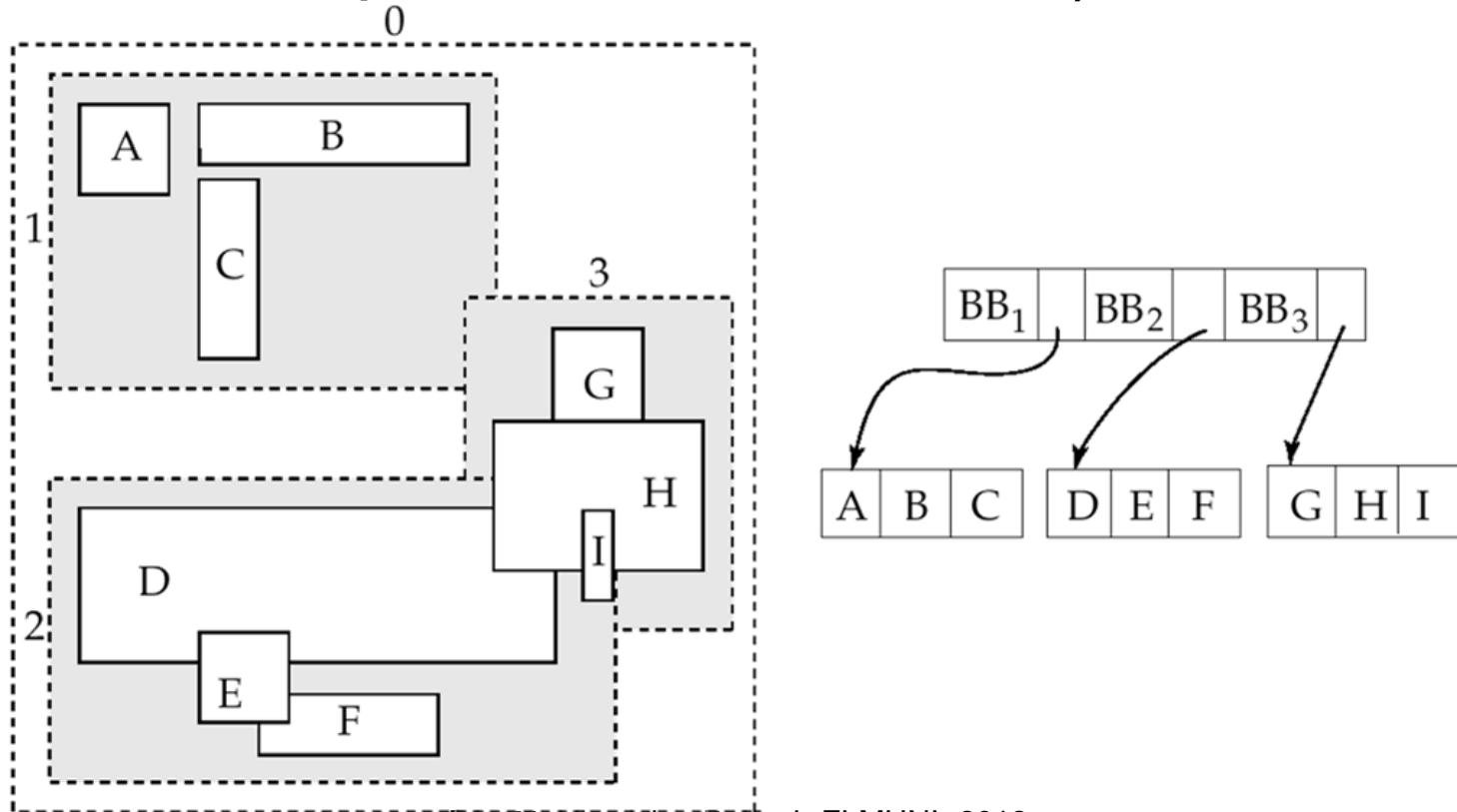
- Rozšíření B<sup>+</sup> stromů na  $d$  rozměrná data
  - Vkládání, mazání je v podstatě stejné
- List obsahuje několik datových prvků
  - List je popsán minimálním pokrývajícím obdélníkem (*minimum bounding rectangle* - MBR)
- Vnitřní uzly odkazují na potomky
  - Každý potomek má MBR
  - Vnitřní uzel je pak popsán novým obdélníkem pokrývajícím všechny potomky
- 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, spuštění

# Přístupová práva

## ■ Databáze

- Obvykle jemnější práva než u souborového systému
- Specifická práva pro
  - tabulky, sekvence, schéma, databáze, procedury, ...
- Pohledy (views)
  - základní nástroj pro řízení kontroly
- 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)
  - 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:

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

# Přístupová práva

- Omezení přístupu pomocí pohledu
  - Relace
    - Zamestnanci(id, jmeno, adresa, plat)
  - Chceme chránit 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

- Přidá se fráze „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.

# Přístupová práva

## ■ Odebírání práv

### □ Přidání za REVOKE

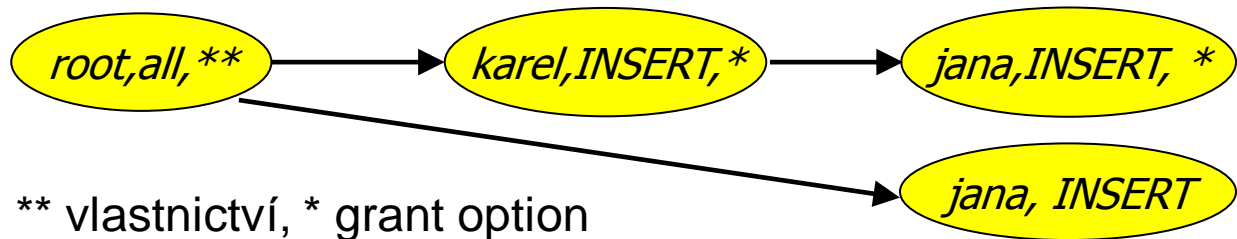
- CASCADE – zruší i oprávnění povolené uživatelem, kterému právě oprávnění odebírám
- RESTRICT (implicitní volba) – odebere pouze toto oprávnění
  - Pokud existují další oprávnění udělená uživatelem, kterému právo odebírám, příkaz skončí chybou.

### □ REVOKE GRANT OPTION FOR ...

- Zruší pouze povolení udělovat oprávnění dalším uživatelům.
- 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 (uživatel / 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 vytvoříme hranu z  $AP^*$  nebo z  $AP^{**}$  do  $BP$ .
  - Nebo do  $BP^*$ , pokud bylo použito „*with grant option*“.
- Když  $A$  uděluje nižší oprávnění  $Q$  než je  $P$ , pak hrana vede do uzlu  $BQ$ .

# Přístupová práva – diagram

## ■ Test oprávnění

- Uživatel  $C$  má oprávnění  $Q$ , pokud existuje cesta z  $OP^{**}$  do  $CQ$ ,  $CQ^*$ ,  $CQ^{**}$  a  $P$  je vyšší oprávnění než nebo stejné jako  $Q$ .
- Poznámka:
  - $P$  může být rovno  $Q$ .
  - $O$  je vlastník a může být  $O = C$ .

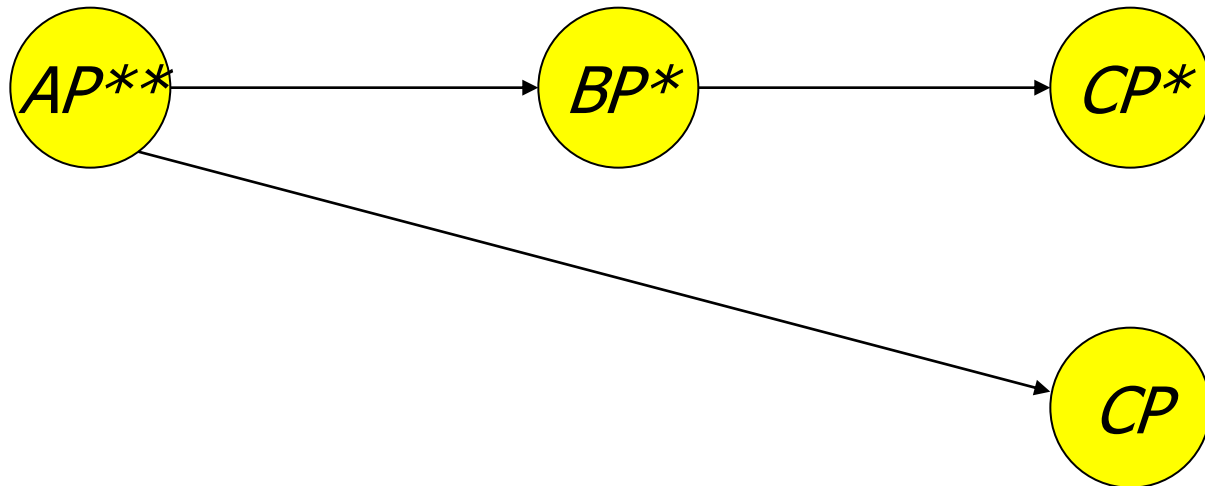


# Přístupová práva – diagram

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

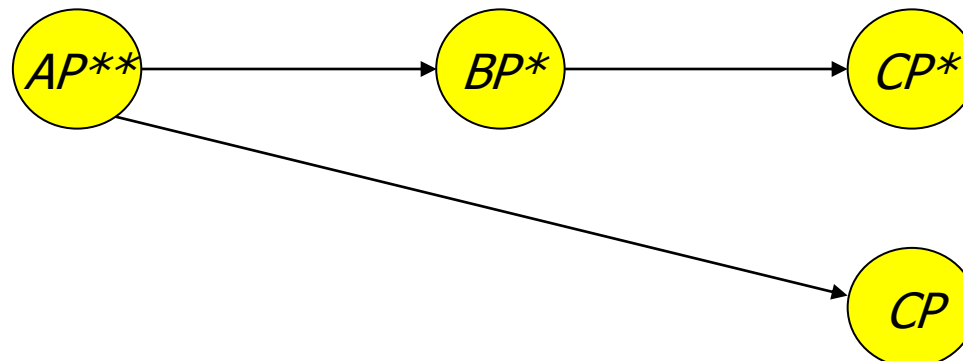
# Přístupová práva – diagram

## ■ 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.



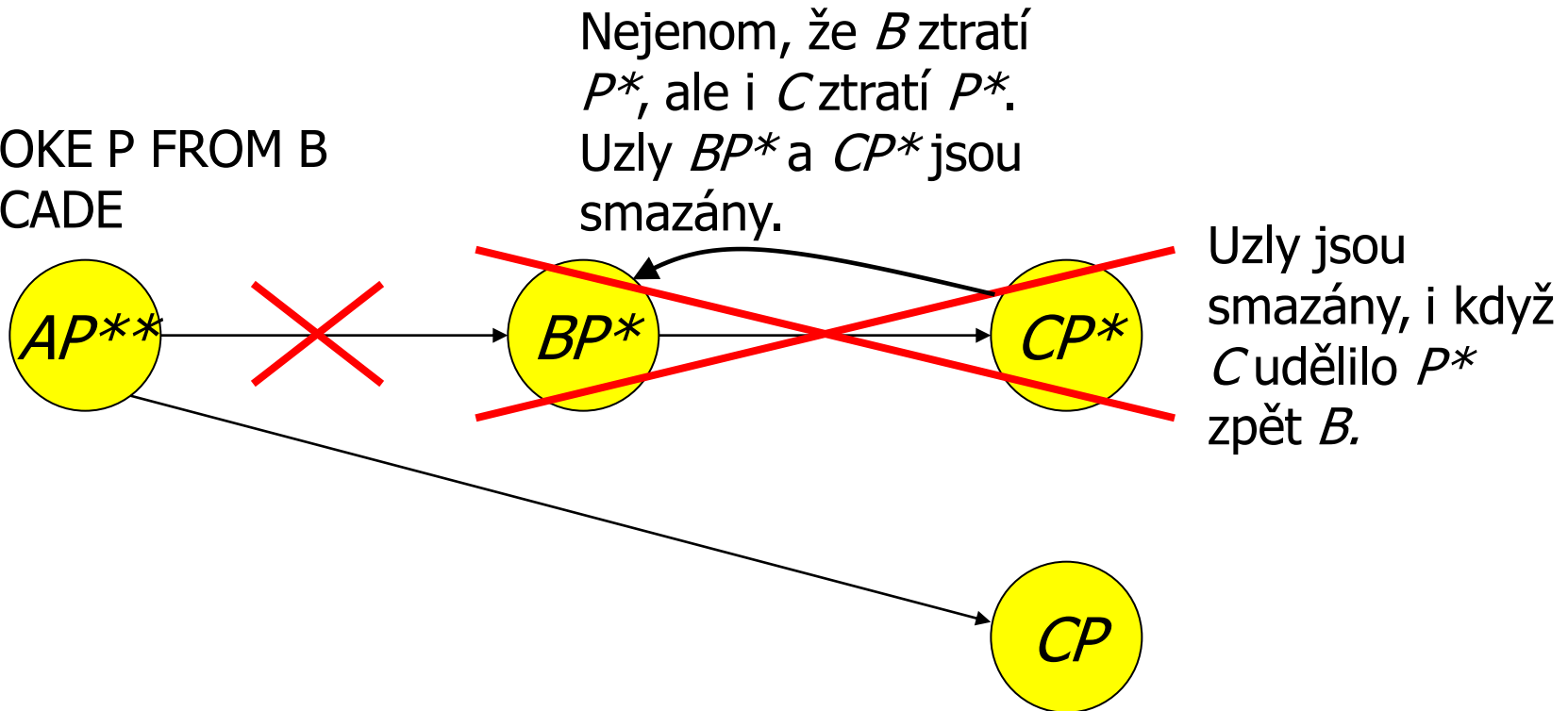
# Přístupová práva – diagram

## ■ 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



Avšak  $C$  bude stále mít oprávnění  $P$ , protože jej získalo i přímo od  $A$ .

# 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, vkládání řádků do různých tabulek, vypočítání průměrného platu, vzdálenost GPS souřadnic, ...
- 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 avgсал() RETURNS real AS 'SELECT avg(plat) FROM zamestnanci' LANGUAGE SQL;
- Uživatel použije pro získání průměru:
  - SELECT avgсал();

# Uložené procedury

## ■ Příklad (pokr.):

- Takové řešení nám ale platy *nezabezpečí*
- Je nutné provést
  - REVOKE SELECT ON Zamestnanci FROM ...
  - GRANT EXECUTE ON FUNCTION avgсал() TO ...
- Při provádění SELECT avgсал(); 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
- Zabezpečené připojení
  - Šifrování 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:
  - Uživatel zadá: „Vader’; DROP TABLE zakaznik; --“
- Aplikace připraví příkaz pro databázi:
  - UPDATE zakaznik SET pozn='\$poznamka' WHERE id=current\_user;
- Po doplnění vstupu se provede:
  - UPDATE zakaznik SET pozn='Vader’;  
DROP TABLE zakaznik;  
--' WHERE id=current\_user;

# 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()* – given string 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

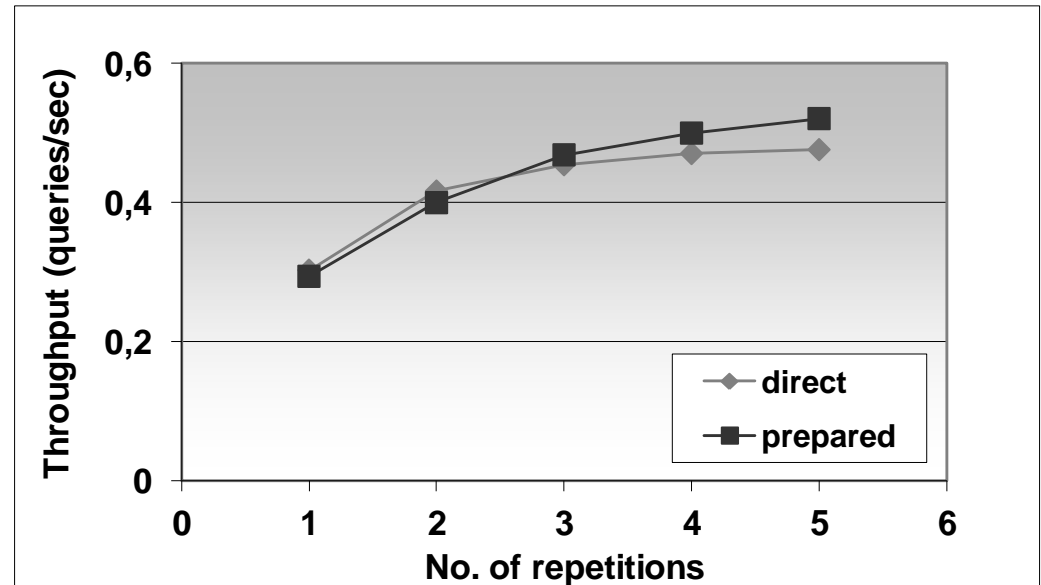
- Prepared statements at server-side
  - The same concept, but stored in DB
  - Typically in procedural languages in DB
- PostgreSQL
  - `PREPARE emp_row(text) AS  
SELECT * FROM emp WHERE name LIKE $1;  
EXECUTE emp_row('%John%');`
- Query is planned in advanced
  - Planning time can be amortized
 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