



PA152: Efektivní využívání DB  
10. Ladění schéma

Vlastislav Dohnal

# Schéma

## ■ Schéma relace

- Seznam atributů, jejich typů a integritních omezení

- Např.

  - Relace student(uco, jmeno, prijmeni, datum\_narozeni)

## ■ Schéma databáze

- Schéma všech relací

# Rozdíly ve schématech

- Stejná data lze uložit různými způsoby

## ■ Příklad

- Dodavatelé

- Adresa

- Objednávky

- Výrobek, počet, dodavatel

# Rozdíly ve schématech

## ■ Alternativy

### □ Schéma 1

- Objednávka1(dodavatel\_id, výrobek\_id, počet, dodavatel\_adresa)

### □ Schéma 2

- Objednávka2(dodavatel\_id, výrobek\_id, počet)
- Dodavatel(id, adresa)

## ■ Rozdíly

### □ Schéma 2 šetří místem

- Schéma 1 nemusí zachovat adresu, pokud není objednávka

# Rozdíly ve schématech

## ■ Rozdíly ve výkonu

- Častý přístup k adrese dodavatele konkrétního výrobku

- → schéma 1 je vhodnější (není potřeba spojení)

- Mnoho objednávek

- → schéma 1 plýtvá místem (opakování adresy)

- → relace bude mít mnoho bloků

# Teorie pro správný návrh schéma

## ■ Normální formy

□ 1NF, 2NF, 3NF, Boyce-Coddova NF, ...

## ■ Funkční závislost

□  $A \rightarrow B$

■ *B funkčně závisí na A*

■ Hodnotu atributu *B* zjistíme, pokud známe hodnotu atributu *A*

■ Necht' *t*, *s* jsou lib. řádky relace,  
pak platí:  $t[A] = s[A] \Rightarrow t[B] = s[B]$

# Teorie pro správný návrh schéma

- Objednávka1 (dodavatel\_id, výrobek\_id, počet, dodavatel\_adresa)
- Příklad funkčních závislostí
  - dodavatel\_id → dodavatel\_adresa
  - dodavatel\_id, výrobek\_id → počet

# Teorie pro správný návrh schéma

## ■ K je primární klíč

- $K \rightarrow R$

- $L \not\rightarrow R$  pro libovolné  $L \subset K$

- Tj. pro každý atribut  $A$  z  $R$  platí:  $K \rightarrow A$  a  $L \not\rightarrow A$

## ■ Příklad

- Dodavatel(id, adresa)

- $id \rightarrow adresa$

- *id* je primární klíč



# Teorie pro správný návrh schéma

## ■ Příklad

- Objednávka1 (dodavatel\_id, výrobek\_id, počet, dodavatel\_adresa)
- dodavatel\_id → dodavatel\_adresa
- dodavatel\_id, výrobek\_id → počet
- *dodavatel\_id, výrobek\_id* je primární klíč

# Normalizace schéma

- 1NF – všechny atributy jsou atomické
- 2NF – všechny atributy závisí na celém klíči
- 3NF – všechny atributy závisí přímo na klíči
  - není tranzitivní závislost
  
- Normalizace = převod do 3NF

# Normalizace schéma

- Relace  $R$  je normalizovaná pokud
  - Pro každou funkční závislost  $X \rightarrow A$  nad atributy relace  $R$  platí, že  $X$  je (super-)klíč.
- Příklad
  - Objednávka1 (dodavatel\_id, výrobek\_id, počet, dodavatel\_adresa)
    - $\text{dodavatel\_id} \rightarrow \text{dodavatel\_adresa}$
    - $\text{dodavatel\_id}, \text{výrobek\_id} \rightarrow \text{počet}$
  - Není normalizovaná

# Normalizace schéma

## ■ Příklad

- Objednávka2(dodavatel\_id, výrobek\_id, počet)
  - dodavatel\_id, výrobek\_id → počet
- Dodavatel(id, adresa)
  - id → adresa
- Schéma je normalizované

# Příklad normalizace

## ■ Banka

- Zákazník má otevřený účet
- Zákazník má korespondenční adresu
- Účet je vedený konkrétní pobočkou banky

## ■ Je relace normalizovaná?

- Banka(zákazník, účet, adresa, pobočka)

# Příklad normalizace

## ■ Relace

- Banka(zákazník, účet, adresa, pobočka)
- zákazník → účet
- zákazník → adresa
- účet → pobočka

## ■ Zákazník je primární klíč

- Důkaz lze provést pomocí funkčních závislostí

## ■ Relace není normalizovaná

- Máme tranzitivní závislost

# Příklad normalizace

- Rozklad relace na
  - Banka(zákazník, účet, adresa)
    - zákazník → účet
    - zákazník → adresa
  - Účet(účet, pobočka)
    - účet → pobočka
  - Nyní již je normalizované

# Postup návrhu schéma

- Identifikace entit

  - Zákazník, dodavatel, objednávka, ...

- Každá entita má atributy

  - Zákazník má adresu, telefon, ...

- Entita musí splňovat:

1. Atribut nemá další atribut (je atomický).

2. Musí existovat funkční závislost pro každý atribut.



# Postup návrhu schéma

- Každá entita je nová relace
- Vztah mezi entitami je vyjádřen novou relací
  - PracujeNa(zaměstnanec\_id, projekt\_id)
- Nalezení funkčních závislostí mezi všemi atributy a kontrola normalizace schéma
  - Pokud existuje závislost  $AB \rightarrow C$ , pak  $ABC$  musí být ve stejné relaci.

# Vertikální dělení

## ■ Příklad: Telefonní operátor

- Entita zákazník má id, adresu a zbývající kredit

- Závislosti:

- id → adresa
- id → kredit

- Normalizované schéma

- Zákazník(id, adresa, kredit)

- Nebo

- ZákazníkAdresa(id, adresa)
- ZákazníkKredit(id, kredit)

- Které je vhodnější?

# Vertikální dělení

- Volba správného schéma závisí na způsobu používání (dotazování)
  - Výpisy z účtu jsou posílány jednou měsíčně.
  - Kredit je měněn po každém telefonním hovoru.
- → Druhé schéma je vhodnější
  - Relace ZákazníkKredit bude menší
    - Méně bloků, menší řídký index
    - Může se vejít do paměti
    - → rychlejší table/index-scan

# Vertikální dělení

- Jedna relace je vhodnější než dvě
  - Pokud jsou atributy přístupovány současně
  - → není potřeba operace spojení
- Dvě relace jsou vhodnější
  - Atributy jsou přístupovány samostatně (nebo některé řádově častěji)
  - Atributy jsou velké (dlouhé řetězce, ...)
    - Pozor: LOB jsou uloženy mimo relace.

# Vertikální dělení

- Jiný příklad

- Zákazník má id a adresu (ulice, město, psč)

- Jaký má smysl schéma:

- ZákazníkUlice(id, ulice)

- ZákazníkMěsto(id, město, psč)

# Vertikální dělení – výkonnost

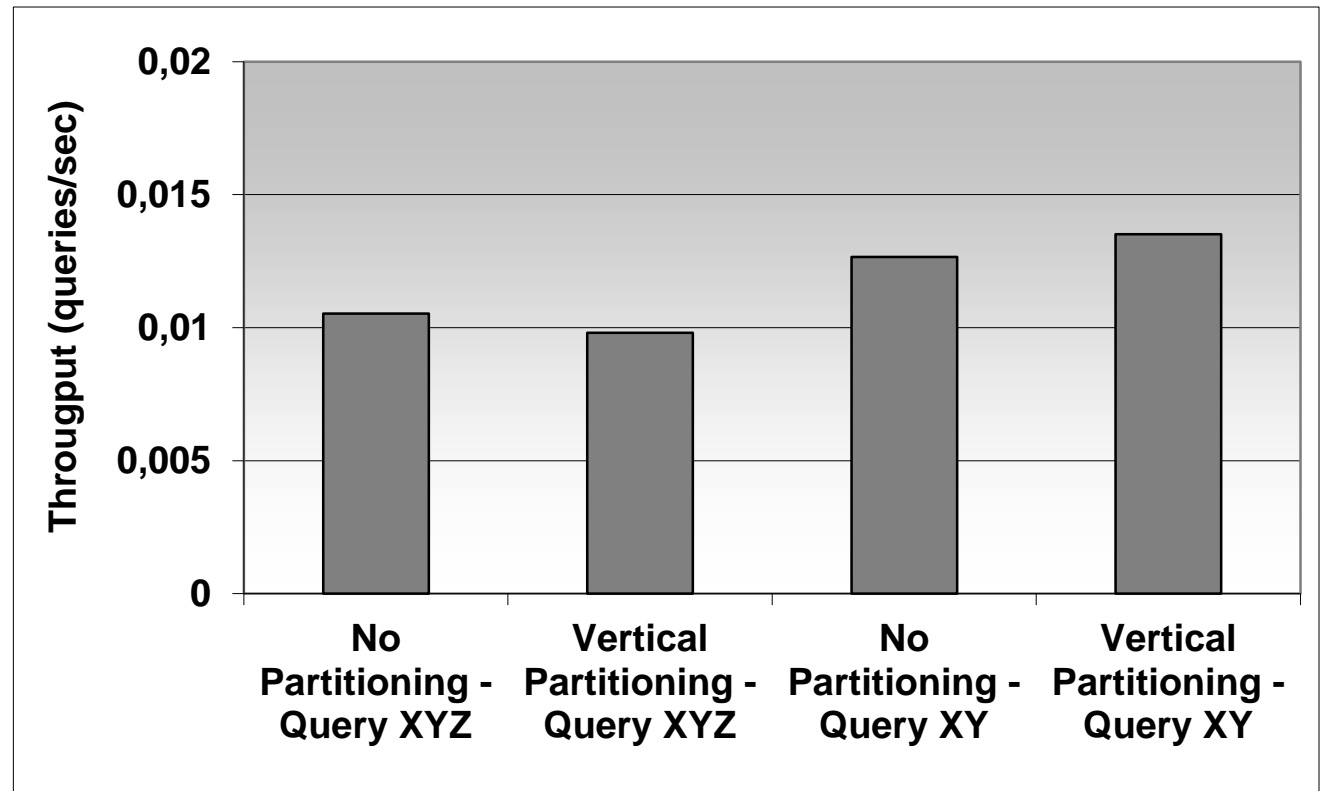
- $R(\underline{X}, Y, Z)$  -  $X$  číslo,  $Y$  a  $Z$  dlouhé řetězce
  - Rychlost závisí na stylu dotazování

## Table-scan

Bez dělení:  
 $R(\underline{X}, Y, Z)$

Vert. dělení:  
 $R1(\underline{X}, Y)$   
 $R2(\underline{X}, Z)$

SQLServer 2k  
Windows 2k



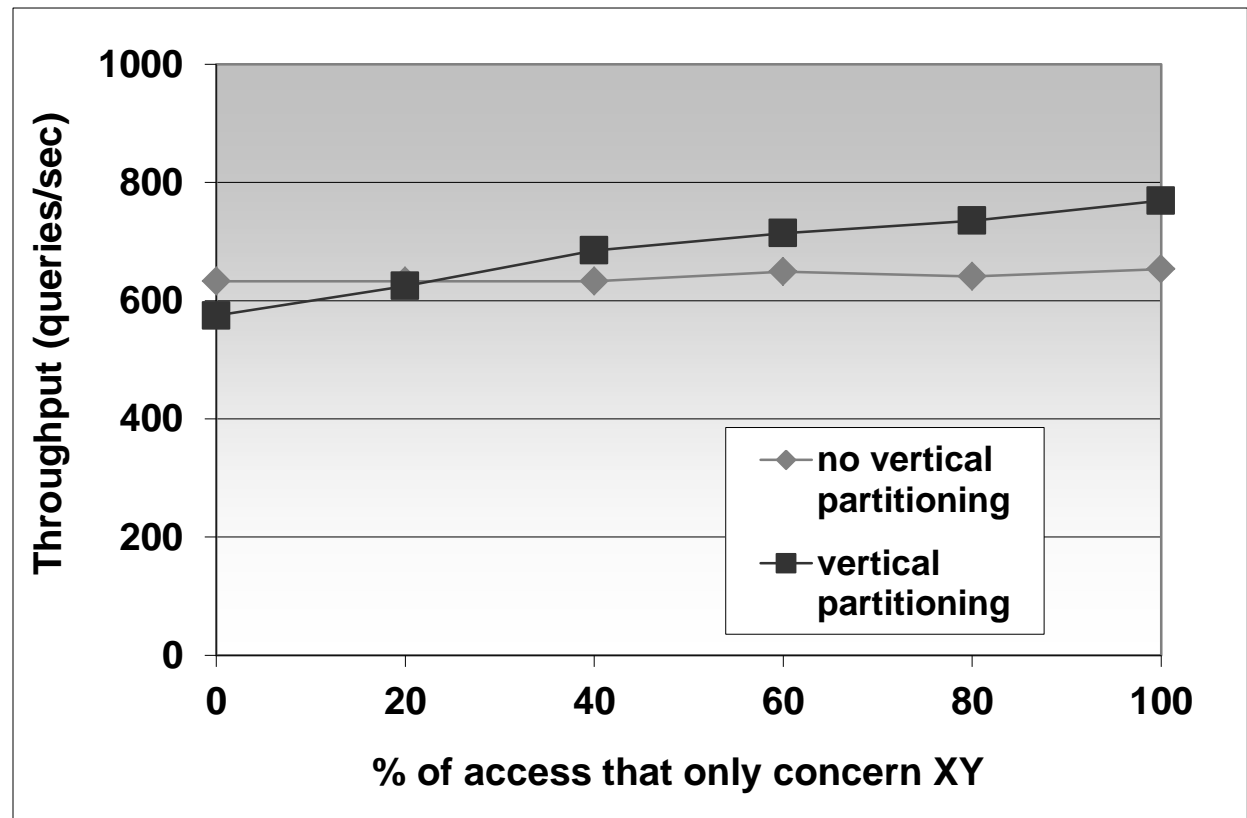
# Vertikální dělení – výkonnost

- $R(\underline{X}, Y, Z)$  -  $X$  číslo,  $Y$  a  $Z$  dlouhé řetězce
  - Výběr 1 řádku, projekce  $XY$  nebo  $XYZ$

## Vert. dělení

Vhodné, pokud vybírám  $XY$  častěji než v 25% případů.

Spojení znamená 2 přístupy do indexu.



# Vertikální spojování (antipartitioning)

- Začínáme s normalizovaným schématem
- Přidáváme atributy k jedné z relací
- Příklad
  - Akciový trh
    - Historie cen akcií za posledních 3 000 dní
    - Makléř se rozhoduje hlavně podle posledních 10 dnů
  - Schéma
    - AkcieDetail(akcie\_id, datum\_vydání, firma)
    - AkcieCena(akcie\_id, datum, cena)



# Vertikální spojování (antipartitioning)

## ■ Schéma

- AkcieDetail(akcie\_id, datum\_vydání, firma)
- AkcieCena(akcie\_id, datum, cena)

## ■ Dotazy na 10ti denní historii jsou náročné

- I když je index na *akcie\_id, datum*
- Navíc pro další informace je třeba spojení s AkcieDetail

# Vertikální spojování (antipartitioning)

- Provedme replikaci dat
- Schéma
  - AkcieDetail(akcie\_id, datum\_vydání, firma, cena\_dnes, cena\_včera, ..., cena\_před\_10\_dny)
  - AkcieCena(akcie\_id, datum, cena)
- Dotazování na 10ti denní historii je
  - 1x prohledání indexu, není třeba spojení

# Vertikální spojování (antipartitioning)

## ■ Nevýhoda

### □ Replikace dat

- Ne příliš velká

- Lze odstranit neukládáním v AkcieCena

- → dotazy na průměrné ceny se komplikují, ...

# Ladění denormalizace

## ■ Denormalizace

- Porušení normalizace schéma
- Pouze z důvodu rychlosti!

## ■ Vhodné pro

- Současný výběr atributů z různých relací

## ■ Nevhodné pro

- Časté aktualizace dat
  - → vyhledání „zdrojových“ dat kvůli replikaci

# Ladění denormalizace

## ■ Příklad

- **region**(r\_regionkey, **r\_name**, r\_comment)
- **nation**(n\_nationkey, n\_name, **n\_regionkey**, n\_comment)
- **supplier**(s\_suppkey, s\_name, s\_address, **s\_nationkey**, s\_phone, s\_acctbal, s\_comment)
- **item**(i\_orderkey, i\_partkey, **i\_suppkey**, i\_linenummer, i\_quantity, i\_extendedprice, i\_discount, i\_tax, i\_returnflag, i\_linestatus, i\_shipdate, i\_commitdate, i\_receiptdate, i\_shipmode, i\_comment)
- $T(\text{item}) = 600\ 000$   
 $T(\text{supplier}) = 500$ ,  $T(\text{nation}) = 25$ ,  $T(\text{region}) = 5$
- Dotaz: vyhledání položek (items) v evropských výrobců

# Ladění denormalizace

## ■ Denormalizovaná relace *item*

- *itemdenormalized* (i\_orderkey, i\_partkey, *i\_suppkey*,  
i\_linenumber, i\_quantity, i\_extendedprice,  
i\_discount, i\_tax, i\_returnflag, i\_linestatus,  
i\_shipdate, i\_commitdate, i\_receiptdate,  
i\_shipmode, i\_comment, ***i\_regionname***);
- 600 000 řádků

# Ladění denormalizace

## ■ Dotazy:

```
SELECT i_orderkey, i_partkey, i_suppkey, i_linenumber,  
       i_quantity, i_extendedprice, i_discount, i_tax,  
       i_returnflag, i_linestatus, i_shipdate, i_commitdate,  
       i_receiptdate, i_shipinstruct, i_shipmode, i_comment, r_name  
FROM item, supplier, nation, region  
WHERE i_suppkey = s_suppkey AND s_nationkey = n_nationkey AND  
       n_regionkey = r_regionkey AND r_name = 'Europe';
```

```
SELECT i_orderkey, i_partkey, i_suppkey, i_linenumber,  
       i_quantity, i_extendedprice, i_discount, i_tax,  
       i_returnflag, i_linestatus, i_shipdate, i_commitdate,  
       i_receiptdate, i_shipinstruct, i_shipmode, i_comment, i_regionname  
FROM itemdenormalized  
WHERE i_regionname = 'Europe';
```

# Ladění denormalizace – výkonnost

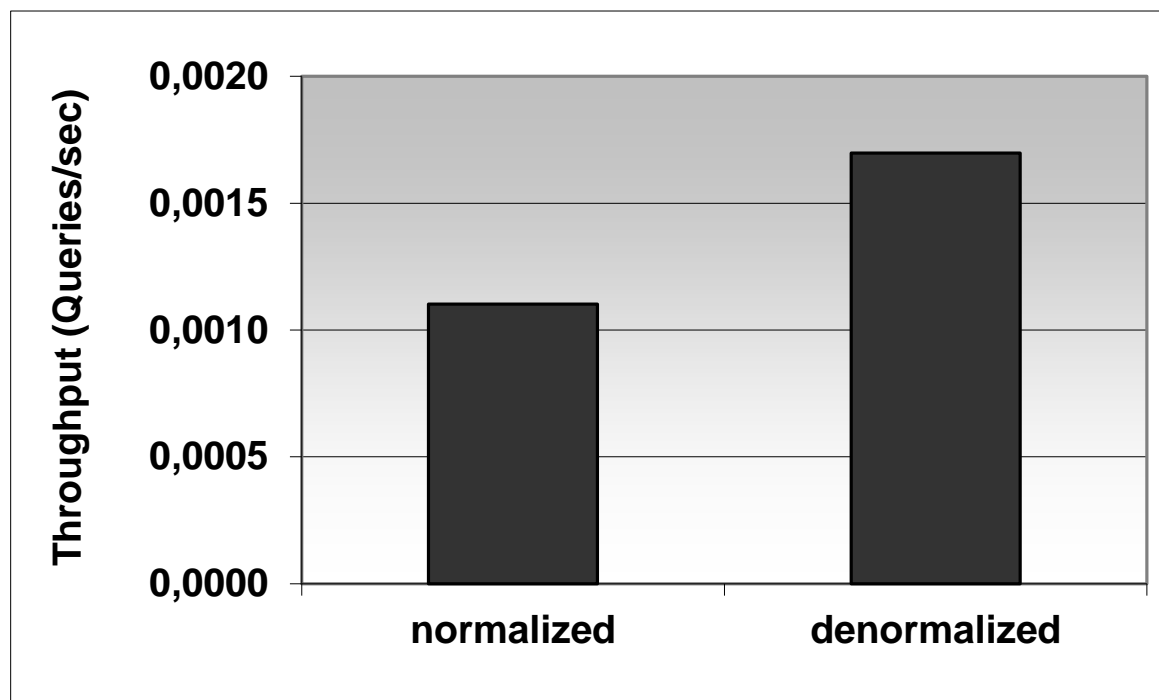
## ■ Dotaz

- Najdi všechny položky evropských výrobců

Normalized:  
spojení 4 relací

Denormalized:  
jediná relace  
54% vylepšení

Oracle 8i EE  
Windows 2k  
3x 18GB disk  
(10 000 ot.)





# Shlukované uložení relací

- Alternativa k denormalizaci
- Není vždy podporováno DB systémem
- Oracle
  - Shlukované uložení dvou relací
    - Objednávka2(dodavatel\_id, výrobek\_id, počet)
    - Dodavatel(id, adresa)
  - Uložení
    - U záznamu dodavatele jsou uloženy jeho objednávky

# Shlukované uložení relací

## ■ Příklad

- Objednávka2(dodavatel\_id, výrobek\_id, počet)
- Dodavatel(id, adresa)

10, Inter-pro.cz Hodonín	12, Školex Modřice
10, 235, 5	12, 12, 50
10, 545, 10	12, 34, 120
11, Unikov Bzenec	
11, 123, 30	
11, 234, 2	
11, 648, 10	
11, 956, 1	

...

# Horizontální dělení

- Rozděluje obsah tabulky podle řádků
  - Vertikální podle sloupců
- Důvod
  - Snížení objemu dat, se kterým se pracuje
  - Usnadnění mazání
- Použití
  - Archivace dat
  - Prostorové dělení
  - ...

# Horizontální dělení

## ■ Automaticky

- Moderní (komerční) DB systémy
  - MS SQL Server 2005 a novější
  - Oracle 9i a novější, ...

## ■ Ručně

- S podporou DB systému
  - Optimalizátor dotazů
- Bez podpory DB systému

# Horizontální dělení

## ■ Změny dotazů:

### □ Automaticky

#### ■ Beze změny

### □ Ručně

#### ■ S podporou DB systému

##### □ Beze změny

##### □ Dědičnost tabulek / definice pohledu (UNION ALL)

#### ■ Bez podpory DB systému

##### □ Ruční změna dotazu

##### □ Je třeba upravit seznam používaných tabulek ve FROM části.

# Horizontální dělení – SQL Server

## ■ MS SQL Server 2005 a novější

### □ Vytvoření dělicí funkce

- CREATE PARTITION FUNCTION
- Dělení na intervaly

### □ Vytvoření dělicího schéma

- CREATE PARTITION SCHEME
- Kam se budou data ukládat (na jaké oddíly úložiště)

### □ Vytvoření dělené tabulky

- CREATE TABLE ... ON dělicí schéma
- Ukládaná data jsou automaticky dělena do oddílů

### □ Vytváření indexů

- CREATE INDEX
- Indexy jsou vytvářeny na oddílech tabulky, tj. automaticky děleny

# Horizontální dělení – Oracle

## ■ Oracle 9i a novější

- Dělení podle rozsahu, výčtu (seznamu), hašování

- Podporuje i dvojité rozdělení

- Oddíly se dělí na pododdíly

- Přímou v CREATE TABLE

[http://docs.oracle.com/cd/B19306\\_01/server.102/b14200/statements\\_7002.htm#i2129707](http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_7002.htm#i2129707)

# Horizontální dělení – PostgreSQL

- PostgreSQL 8.2 a vyšší
  - Dělení podle rozsahu, výčtu (seznamu)
- Princip (<http://www.postgresql.org/docs/current/static/ddl-partitioning.html>)
  - Využití dědičnosti tabulek
    - Vytvoření základní tabulky
      - Nebude ukládat data, bez indexů, ...
    - Jednotlivé oddíly budou zděděné tabulky
      - Pro každou tabulku definovat CHECK omezení povolených dat
    - Vytvoření případných indexů
  - Nevýhoda: nelze používat cizí klíče.



# Horizontální dělení – PostgreSQL

## ■ Princip

### □ Vkládání záznamů

#### ■ Vkládání do primární tabulky

#### ■ Primární tabulka má pravidla pro vkládání

- Vkládání pouze do „nejnovějšího“ oddílu → jeden RULE
- Obecně je třeba mít RULE pro každý oddíl
- Lze realizovat i triggerem...

### □ V případě použití pohledů

#### ■ Lze definovat *INSTEAD OF* trigger

# Horizontální dělení – PostgreSQL

## ■ Příklad v db.fi.muni.cz, schéma xdohnal

### □ Nerozdělená tabulka *account*

#### ■ Primární klíč *id*

#### ■ $R(\text{account}) = 200\ 000$

#### ■ $V(\text{account}, \text{home\_city}) = 5$

<u>home_city</u>	<u>count</u>
home_city1	40020
home_city2	40186
home_city3	39836
home_city4	39959
home_city5	39999

### □ Rozdělená tabulka *account\_parted*

#### ■ Podle *home\_city* (5 oddílů)

#### □ Oddíly *account\_parted1* .. *account\_parted5*

# Horizontální dělení – PostgreSQL

## ■ Statistiky

Tabulka	Řádků	Velikost	Indexy
account	200 000	41 984 kB	4 408 kB
account_parted	0	0 kB	8 kB
account_parted1	40 020	8 432 kB	896 kB
account_parted2	40 186	8 464 kB	896 kB
account_parted3	39 836	8 392 kB	888 kB
account_parted4	39 959	8 416 kB	896 kB
account_parted5	39 999	8 424 kB	896 kB
Celkem:	200 000	42 128 kB	4 472 kB

# Horizontální dělení – PostgreSQL

## ■ Optimalizátor dotazů

- Povolení kontroly omezení na oddílech

```
set constraint_exclusion=on;
```

## ■ Dotazy (porovnejte plány provádění)

```
select * from account where id=8;
```

```
select * from account_parted where id=8;
```

```
select count(*) from account where home_city='home_city1';
```

```
select count(*) from account_parted where home_city='home_city1';
```

```
select * from account where home_city='home_city1' and id=8;
```

```
select * from account_parted where home_city='home_city1' and id=8;
```