Relačná databáza - pojmy

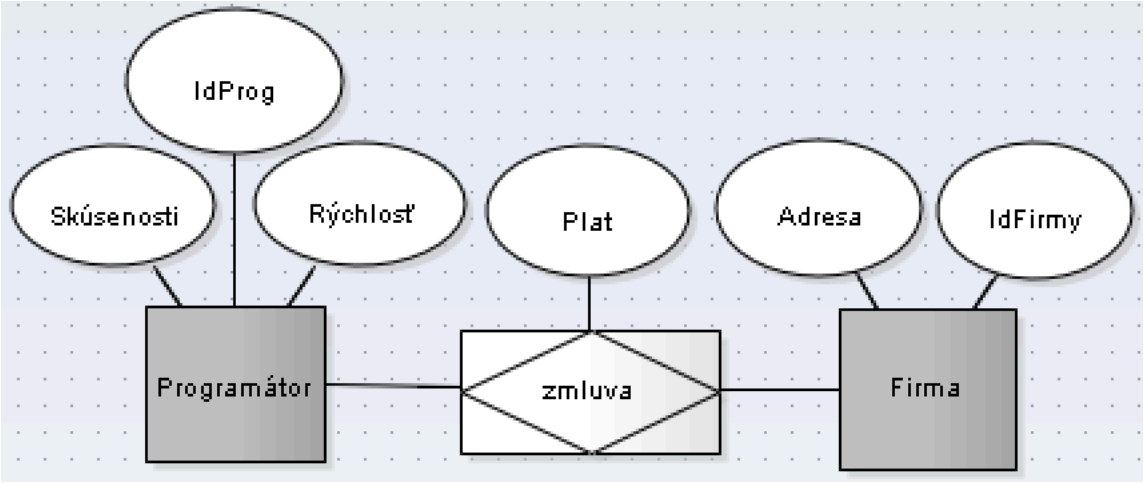
- **Relácia** stupňa n (n-árna relácia) s **kardinalitou relácie** k, je množina k n-tíc, ktorá je podmnožinou karteziánskeho súčinu $R \subseteq D_1 \times D_2 \ldots \times D_n$, kde D_i- doména (množina hodnôt) atribútu i. Dá sa predstaviť ako tabuľka s k riadkami a n stĺpcami (stĺpce sú jednotlivé atribúty)
  - Pozn: Karteziánsky súčin množín A a B, je množina všetkých usporiadaných dvojíc (a, b), kde $a \in A$ a $b \in B$, t.j. platí $A \times B = \{(a, b); a \in A \ \wedge b \in B\}$
  - *Príklad: Majme reláciu s kardinalitou 2, teda 2 atribúty::*
    - *ATR1 = farba, Jeho doména = napr. {červená, zelena, hneda, žltá}*
    - *ATR2 = rola, doména je napr. {učiteľ, študent, opravár}*
  - *Kartézsky súčin týchto 2 množín je množina s 12 prvkami {(červená, učiteľ), (červená, študent),...,}*
- **relačná schéma** je zápis relácie v tvare: Entita (atribút1, atribút2, .., atribút n)
  - napr: študent(osobné číslo, meno, priezvisko, výška, váha)
- n-tica = riadok = záznam = inštancia ; stĺpec = atribút
  - atribúty, ktoré pomenúvavajú jednu doménu (jeden stĺpec) sa nazývajú jednoduché, ich kombinácie sa nazývajú zložené atribúty.
- **Databáza (DB schéma)** je konečná množina relácií, ktoré sú definované nad doménami D
- **Superkľúč relácie R** je množina atribútov relácie R s vlastnosťou **Jednoznačnej identifikácia** – každá n-tica relácie R je hodnotami atribútov, ktoré tvoria superkľúč jednoznačne určená, napr. pre študenta jú superkľúče, napr. (osobné číslo, meno), (osobné číslo, priezvisko), (osobné číslo)
- **Kľúč** je minimálny superkľúč, t.j. naviac spĺňa podmienku **neredundantnosti** – ak sa vynechá nejaký atribút z kľúča K poruší sa jednoznačnosť, z vyššieuvedených superkľúcov to je (osobné číslo)
- **Kandidát na primárny kľúč** = kľúč. Vybraný kľúč relácie R sa označí ako **primárny kľúč (PK)** (bude sa podtrhovať), ostatné sú Alternatívne kľúče **(AK)**
  - Príklad na PK: študent(osobné číslo, meno, priezvisko, výška, váha)
- **cudzí kľúč** (foreign key) (FK) – kľúč, ktorý je v inej tabuľke primárnym kľúčom (podtrhuje sa čiarkovane), napr. učiteľ (meno, škola)
- Vzťah medzi entitami v ER diagrame sa reprezentuje pomocou **relácie**, ktorá obsahuje kľúčové atribúty entít ktoré prepája

13 pravidiel pre relačné databázy (Dr. E.F. Codd 1970)

1) DBS musí byť schopný spravovať DB **len pomocou svojich relačných schopností**
2) Všetky informácie v DBS (vrátane názvov tabuliek a stĺpcov) sú reprezentované explicitne ako **hodnoty v tabuľke**
3) Ku každej hodnote uloženej v DBS sa musí dať pristúpiť pomocou:
   a. **názvu tabuľky + primárneho kľúča + názvu atribútu**
4) DBS poskytuje podporu pre **prácu s hodnotou** $\boldsymbol{null}$ (neznáme, nedefinované, mimo rozsah, chybné údaje), ktoré sú odlišné od hodnôt v akejkoľvek doméne
5) Popis databázy a jej obsahu je reprezentovaný **na logickej úrovni v tabuľkovej forme** nad ktorými sa dá dopytovať pomocou databázového jazyka
6) **Databázový jazyk musí mať dobre definovanú syntax** a musí byť ucelený. Musí podporovať definíciu údajov, manipuláciu s údajmi, pravidlá integrity, autorizáciu, transakcie.
7) **Pohľady (views)** musia byť aktualizovateľné prostredníctvom DBS
8) DBS podporuje **operácie na úrovni množín** (získavanie údajov, vkladanie, aktualizáciu, ...)
9) **Fyzická dátová nezávislosť** – zmena fyzických štruktúr nemá dopad na logické
10) **Logická dátová nezávislosť** - zmena tabuľkových štruktúr nemá mať vplyv na aplikačné programy a AHQ (ad hoc query)
11) DBS musí byť schopný **definovať pravidlá integrity**. Tieto musia byť dostupné online
12) **Distribúcia, resp. redistribúcia** údajov nemá na aplikačné programy/AHQ žiadny vplyv.
13) **Nesmie** existovať možnosť **obísť pravidlá integrity**

Príklad 1 – ER diagram a príklad jeho tabuľkovej formy

vyjadrenie väzby medzi entitami reláciou



Databáza bude obsahovať relačné schémy:

- Programátor(IdProg, Skúsenosti, Rýchlosť)
- Firma(IdFirmy , adresa)
- Zmluva(IdProg , IdFirmy, Plat) platí síce aj Zmluva(IdProg , IdFirmy, Plat)
    - (medzi programátorom a firmou ale môže byť iba jedna zmluva)

Resp. pomocou zavadenia idZmluvy (medzi programátorom a firmou môže byť viac zmluv)

- Programátor(IdProg, Skúsenosti, Rýchlosť)
- Firma(IdFirmy , adresa)
- Zmluva(idZmluvy, IdProg , IdFirmy, Plat, DefiniciaPlnenia)

Toto je nevhodné prečo? (údaje sú uložené efektívne iba keď programátori a firmy majú po jednej zmluve, v opačnom prípade sa začnú kopiť opakované atribúty)

- Programátor(IdProg, Skúsenosti, Rýchlosť, IdZmluvy)
- Firma(IdFirmy , adresa, idZmluvy)
- Zmluva(idZmluvy, IdProg , IdFirmy, Plat), resp. stačí Zmluva(idZmluvy, Plat, DefiniciaPlnenia)

Príklad 2

Jednodúché previazanie tabuliek pomocou cudzieho kľúča:

- Zamestnanec (ID, meno, pozícia, ID odddelenia)
- Oddelenie (ID oddelenia, názov, vedúci)

Príklad3:

Porovnajme schémy:

Zamestnanec (ID, meno, pozícia, ID odddelenia) Oddelenie (ID oddelenia, názov, vedúci)	ZamestnanecFull (ID, meno, pozícia, ID oddelenia, názov, vedúci)

Prečo je schéma na pravo problematická?

- V riadkoch tabuľky (relačnej schémy) ZamestnanecFull **sú opakované nadbytočné informácie** – ak sa opakuje ID oddelenia, zároveň sa opakuje názov a vedúci oddelenia.

Anomálie

- Môžu vznikať chybným návrhom databázy (štrukturálne vady databázy)
- Anomália vloženia
  - Čo ak do tabuľky ZamestnanecFull vložíme 2 záznamy, ktoré budú mať to isté ID oddelenia, ale iný názov a vedúceho?
- Anomália odstránenia
  - Čo ak z tabuľky ZamestnanecFull odstránime všetkých zamestnancov, ktorí majú zvolené ID oddelenia. V schéme neostane potom žiadny záznam, že takéto oddelenie vôbec existuje.
- Anomália aktualizácie
  - Čo ak sa zmení vedúci nejakého oddelenia, ak to v tabuľke ZamestnanecFull u niektorých zamestnancov z tohto oddelenia zmeníme a u niektorých na to zabudneme?

Príklad4:

Ak nejakí zamestnanci (a nie je ich veľa) sú niečím výnimoční, pridaj k nim pole poznámka. Ako na to? Ktorý z nižšie uvedených návrhov je lepší?

Zamestnanec (ID, meno, pozícia, IDodddelenia, Poznámka)	Zamestnanec (ID, meno, pozícia, IDodddelenia) Poznámka(IDzamestnanca, Poznámka)
(Tento nie)	(Tento áno, prečo?) lebo vytvárame poznámku iba v prípade potreby

Prečo je tento návrh najlepší?

Zamestnanec (ID, meno, pozícia, IDodddelenia)
Poznámka(IDpoznamky, IDzamestnanca, Poznámka)
(umožňuje vytvoriť viacero poznámok aj pre jedného zamestnanca a mať primárny kľúč)

Poďme toto trochu sformalizovať, zaveďme formy a ukážme ich výhody a nevýhody ...

Normalizácia a normálne formy

- normalizácia je proces, pomocou ktorého sa dá databáza zbaviť štrukturálnych vád
- normalizácie je súhrnom niekoľkých tzv. normálnych foriem - množín pravidiel, ktoré hovoria aká by mala byť štruktúra tabuliek
- Prvá až tretia normálna forma sú založené na funkčných závislostiach
- **Funkčná závislosť**:
  - Definícia 1: funkčná závislosť v tabuľke medzi stĺpcom A a B znamená, že hodnota v stĺpci A určuje hodnotu v stĺpci B. Píšeme: $A \rightarrow B$. T.j. pri rovnakých hodnotách v stĺpci A nesmú byť v stĺpci B rôzne hodnoty.
  - Definícia 2: Nech A, B sú dve množiny atribútov, podmnožiny relačnej schémy R(A_1, A_2, ..., A_n), kde A_i sú atribúty. Potom funkčná závislosť A $\rightarrow$ B medzi dvomi množinami atribútov A a B znamená, že ak pre dva záznamy (inštancie) t_1 a t_2 v R platí $t_1[A] = t_2[A]$ potom musí platiť $t_1[B] = t_2[B]$.
    - Pozn. Zápis $t_1[A]$ znamená hodnota atribútov A v inštancii t_1.
- **Silná funkčná závislosť:**
  - Nech A a B sú množiny atribútov relácie R. B je silne funkčne závislá na A, ak B funkčne závisí od celej A a nezávisí od žiadnej podmnožiny A.
- príklad **nenormalizovanej formy**:

Tab.1:

ID_OBJEDNAVKY	**KOD_TOVARU**	**POCET**	**DODAVATEL**		
			KOD	**MENO**	**ADRESA**
O1	100	2	D1	Bača	Koliba
O1	102	17	D2	Fero	Pevnosť

Prvá normálna forma (1NF)

- Hodnoty atribútov resp. stĺpcov musia byť **atomické.** Nesmú byť
  - ani **zložené** (viď predchádzajúca tabuľka)
  - ani **viachodnotové**, napr. nemali by sme mať atribút „jazykové znalosti“, kde by boli pri rôznych osobách rôzne zoznamy ako „angličtina, nemčina, francúzština“, „španielčina, ruština“, ...
  - ani kombinácia uvedených prípadov
- Príklad 1NF:

ID_OBJEDNAVKY	**KOD_TOVARU**	**POCET**	**KOD_D**	**MENO_D**	**ADRESA_D**
O1	100	2	D1	Bača	Koliba
O1	102	17	D2	Fero	Pevnosť

- Ak chceme napr. tabuľku študent (ID, Meno, Jazykové znalosti) previesť do 1NF, máme viac možností
  - A: študent (ID, Meno, Znalosť) – budú sa nám ale opakovať záznamy zdieľajúce to isté ID, meno (ID sa nemôže použiť ako primárny kľúč)
    Tab2: Študent

ID	Meno	Znalosť
1	Jano	Angličtina
1	Jano	Ruština
2	Fero	Angličtina

  - B: vytvoríme dve tabuľky – bez redundancie
    - Študent (<u>ID</u>, meno)
    - znalosť (<u>ID</u>, jazyková znalosť) – ID je cudzí kľúč do tabuľky študent

Druhá normálna forma (2NF)

- všetky atribúty, ktoré nie sú súčasťou primárneho kľúča sú **silne funkčne závislé** na primárnom kľúči a schéma je pritom v prvej normálnej forme
  - ak primárny kľúč tvorí viac stĺpcov každý ďalší atribút v tabuľke musí byť závislý na kombinácii týchto stĺpcov
- Príklad:
  - Tab2. Študent pred dvomi stranami nebola v 2NF:
    - Ak sa zoberie forma študent (ID, Meno, Znalosť), zdá sa že spĺňa požiadavku, t.j. platí
      - ID, Znalosť → Meno
    - Problém je, že platí už aj
      - ID→Meno
    - T.j. je tu závislosť od podmnožiny ! (t.j. sa nejedná o silnú funkčnú závislosť!)
  - Ako dať schému do 2NF?
    - Riešenie: tabuľku treba rozdeliť tak, aby bola prítomná silná funkčná závislosť
      - Študent (ID, meno)
      - znalosť (ID, jazyková znalosť) – ID je cudzí kľúč do tabuľky študent
    - toto riešenie je v 1NF aj 2NF

Tretia normálna forma (3NF)

- pre 2NF bolo ťažiskové, že funkčná závislosť musela byť na CELOM KĽÚČI
- 3NF hovorí, že závislosť nesmie byť na ničom inom, iba na kľúči
  - treba odstrániť ďalšie závislosti.
- Príklad
  - Mali sme príklad 3: ZamestnanecFull (ID, meno, pozícia, ID oddelenia, názov, vedúci)
  - Tento zápis je v 2NF
  - Máme tu závislost:
    - ID→meno, pozícia, ID oddelenia, názov, vedúci
  - Ale aj:
    - ID oddelenia → názov, vedúci
  - Aby to bolo v 3NF, treba druhú závislosť odstrániť
  - Schéma v 3NF bude
    - Zamestnanec (ID, meno, pozícia, ID odddelenia)
    - Oddelenie (ID oddelenia, názov, vedúci)
- 3NF je dostatočná forma na odstránenie redundancie údajov.
- **BCNF forma (Boyce-Coddova NF)**
  - variant 3NF, taký, že pre každú funkčnú závislosť v každej tabuľke je ľavá strana funkčnej závislosti superkľúč.

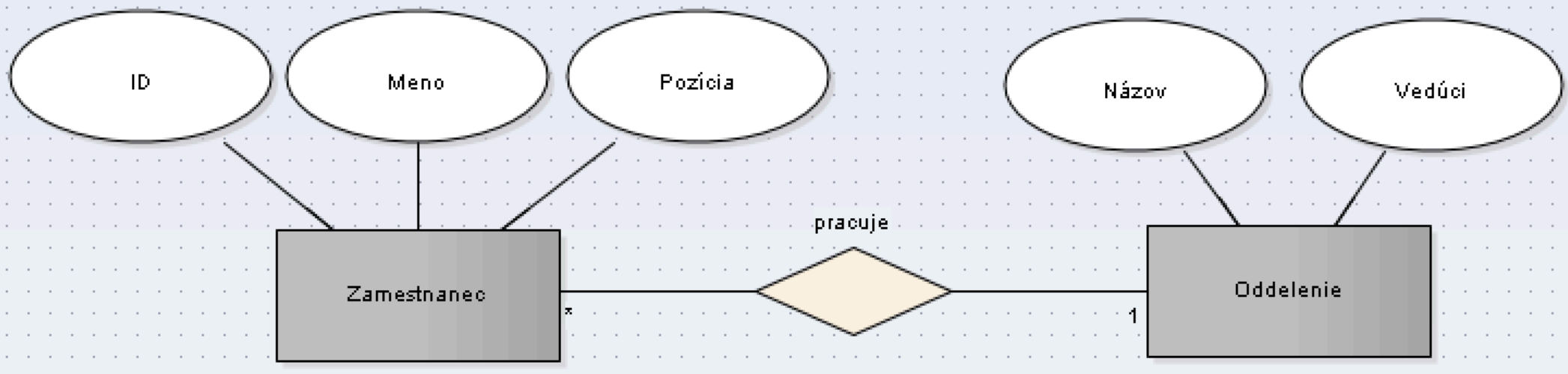
Vyššie normálne formy (4NF, 5NF)

- Existujú, nebudeme s animi zaoberať, 1NF, 2NF, 3NF ako základ stačí

Návrh údajového modelu

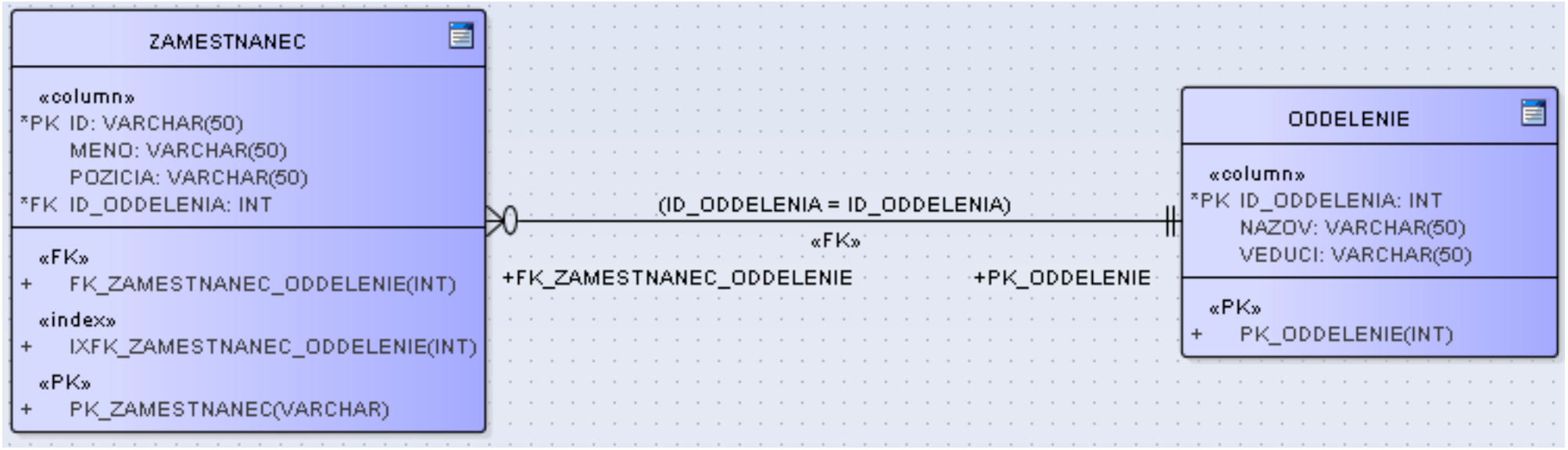
- 3 úrovne
  - koncepčný model (E-R diagram) – entity, relácie, kardinality, atribúty
  - logický model – zväčša UML diagram + typy atribútov
  - fyzický model - tabuľky, PK, FK, podmienky

Koncepčný diagram:



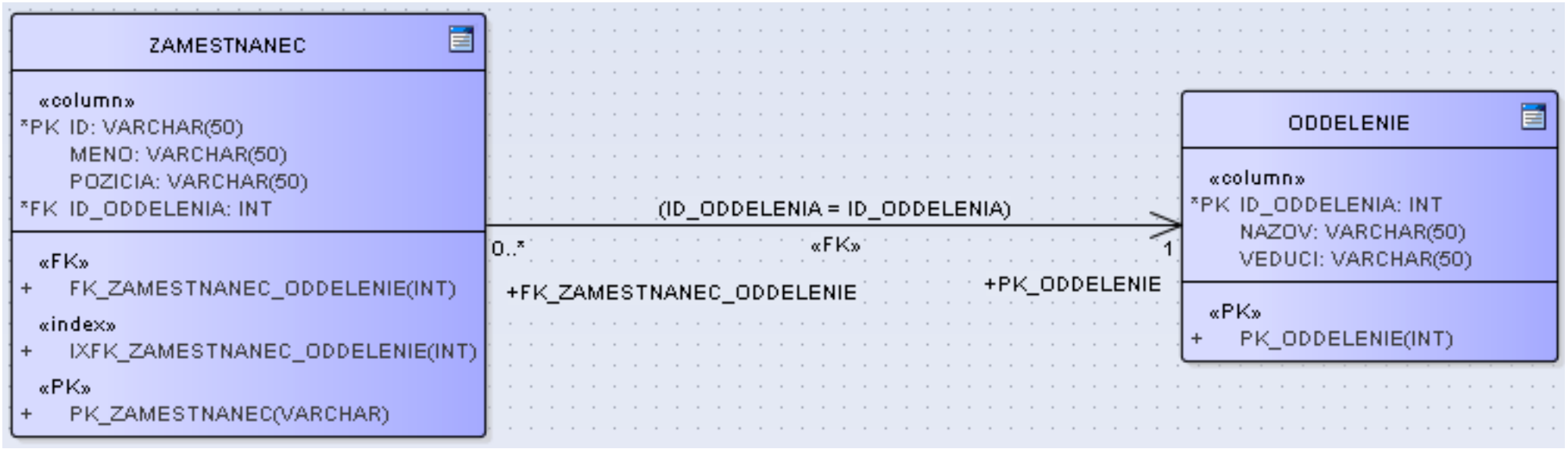
Logický model:

Fyzický model:



Pozn: nad FK sa vytvoril aj “index”

(UML 2.1 notácia)

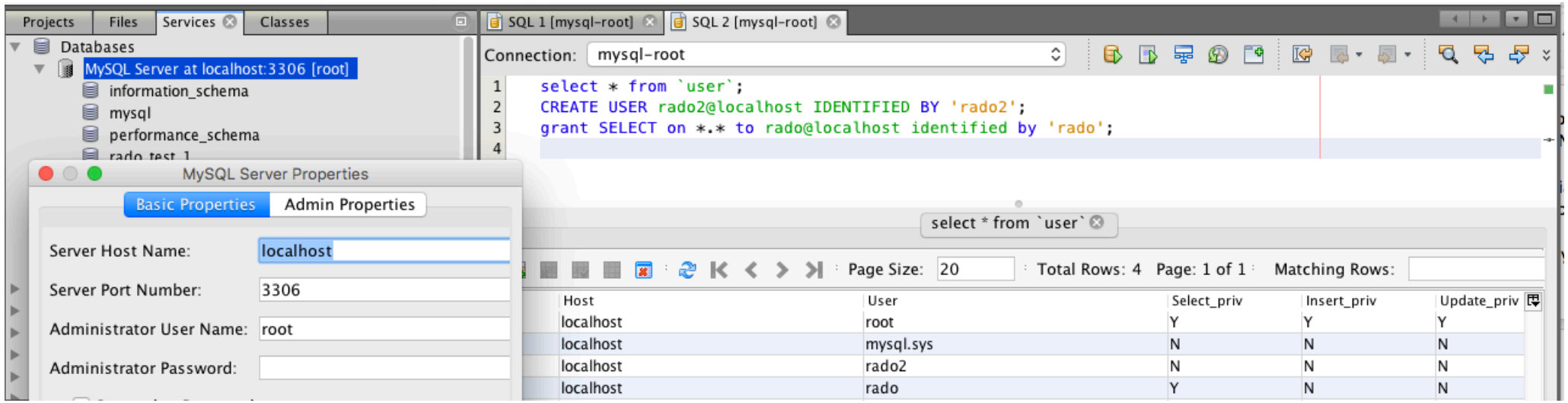


SQL

- Východiská
  - máme navrhnutý údajový model
  - máme nainštalovaný SRDB systém
- Chceme
  - Vytvoriť schému
  - Naplniť tabuľky údajmi
  - pracovať s údajmi
- Urobíme to pomocou jazyka SQL (Structured Query Language) ktorý tvoria dve sémanticky oddelené časti:
  - DDL časť – pre tvorbu databázových štruktúr
  - DML časť – pre prácu s údajmi (vkladanie, mazanie, modifikáciu, dopytovanie)
- Základné fakty o jazyku SQL
  - **Deklaratívny jazyk** (je popísaný želaný výsledok bez špecifikovania postupu po krokoch, ktorý k výsledku má viesť)
  - vyvinutý v 70 rokoch 20. Storočia v IBM labs pre DB2
  - štandardizovaný ako priemyselný štandard ANSI (American National Standards Organisation) a ISO (International Standard Organization)
  - väčšina SRDB obsahuje prorpietárne rozšírenia, ktoré z jazyka robia **procedurálny jazyk**
    - napr. PL/SQL (Oracle), Transact-SQL (Microsoft), ...
  - **Nerozlišuje veľkosť písmen pri príkazoch**
  - Dynamické/statické SQL: budeme používať dynamické – pomocou knižníc (JDBC) a tvorby volaní z reťazcov (→ pozor, toto je postup citlivší na tzv. SQL injection útoky)

MYSQL

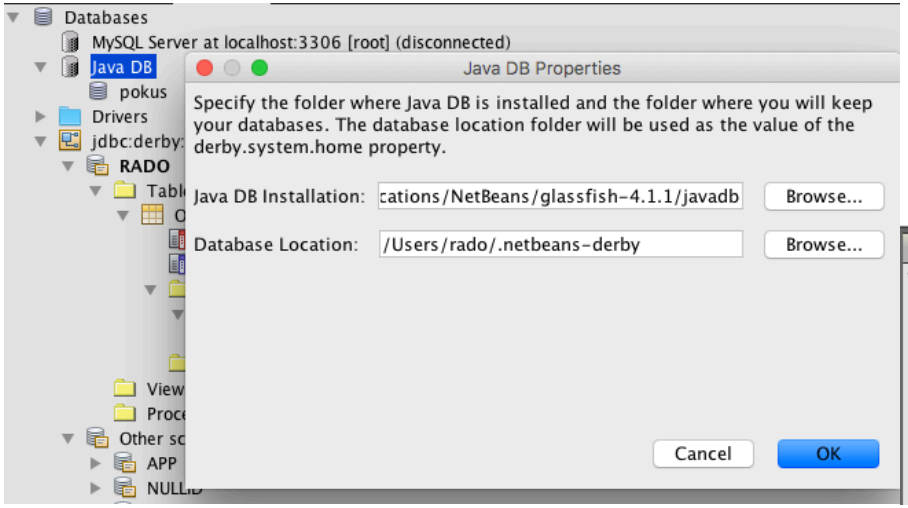
- populárna verzia RDBMS (http://www.mysql.com)
- súčasťou populárnych modelov riešení LA**M**P, MA**M**P
- v súčasnosti máme MySQL 5.7.17
- súčasť distribúcie sú rôzne nástroje z pre príkazový riadok**: mysql** – hlavný prístup z príkazového riadku; **mysqladmin** – admin prístup; **mysqldump** – zálohovanie DB, ...
  - prihlásenie: “mysql -u <user> -p” + zadat heslo ...
  - následne napr: show databases; use <database xy>; show tables; describe tableXY;
  - práca s predpripravenou sadou príkazov: „mysql –u uzivatel -p < subor_s_prikazmi“
- Netbeans ponúka pohodlný prístup priamo z IDE:

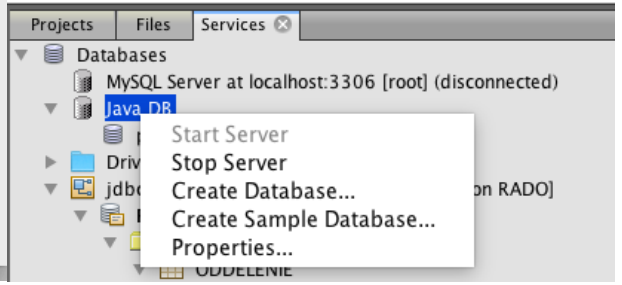


- základné úkony:
  - vytvorenie užívateľov (https://dev.mysql.com/doc/refman/5.7/en/create-user.html)
  - pridelenie prístupov (https://dev.mysql.com/doc/refman/5.7/en/grant.html)
  - ...

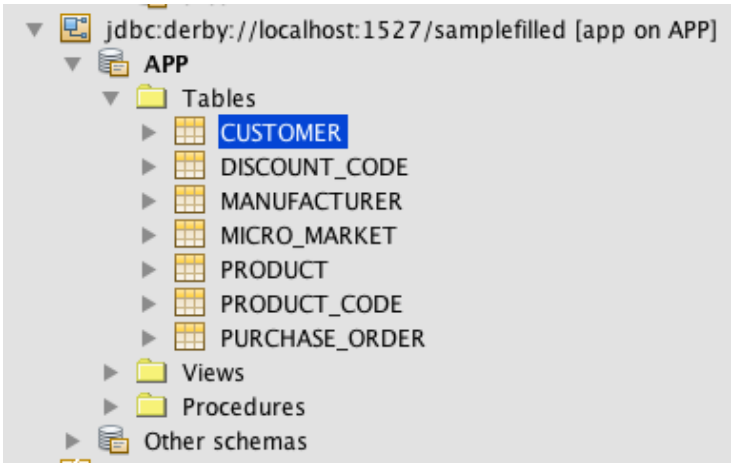
DERBY

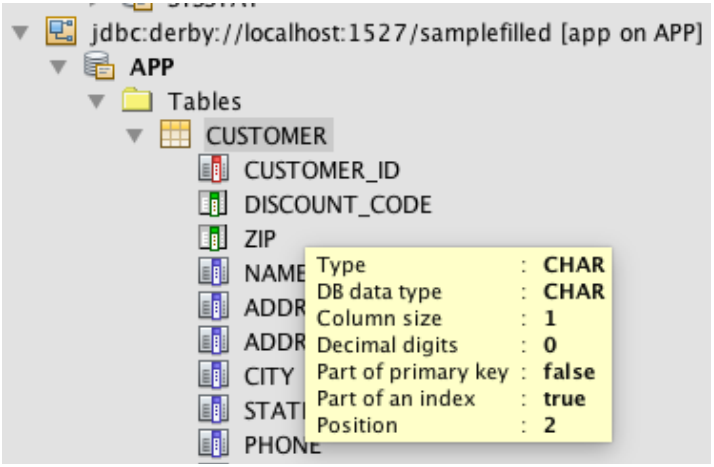
Jednoduchá databáza integrovaná v NetBeans





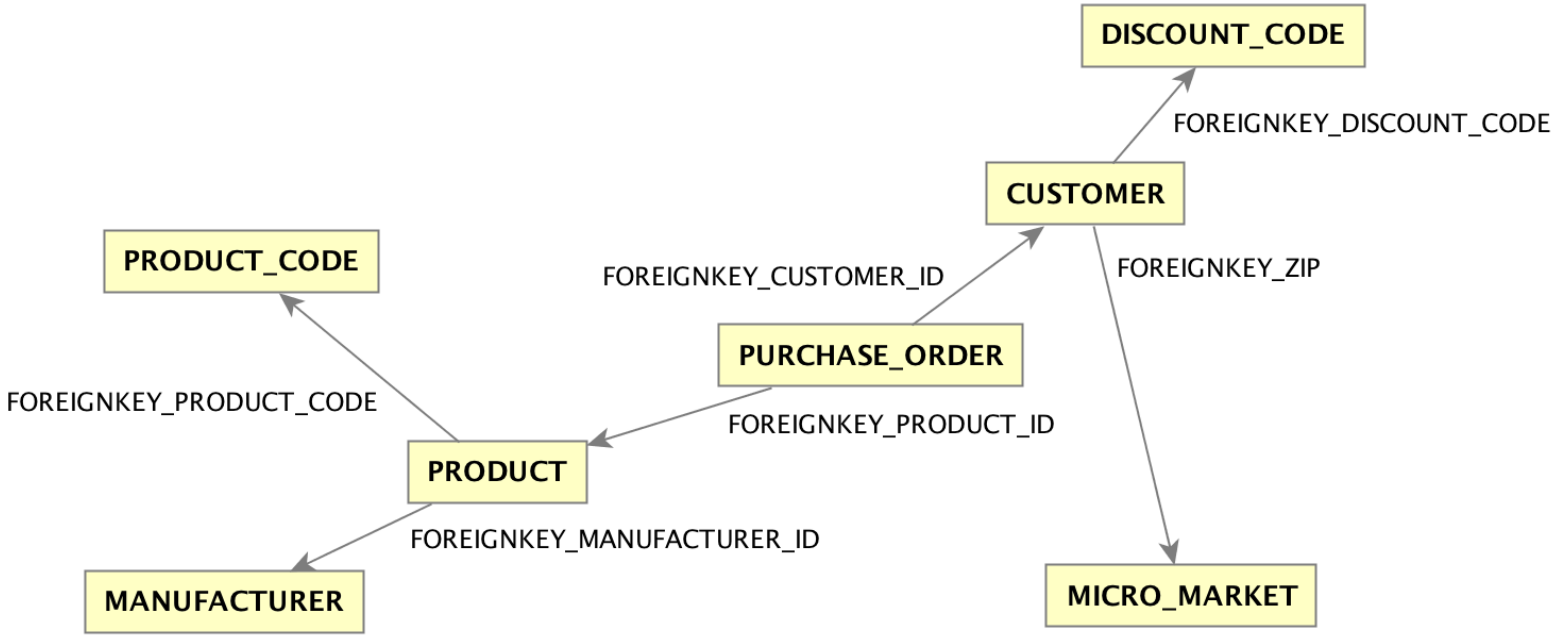
Vzorová databáza (Create Sample Database)



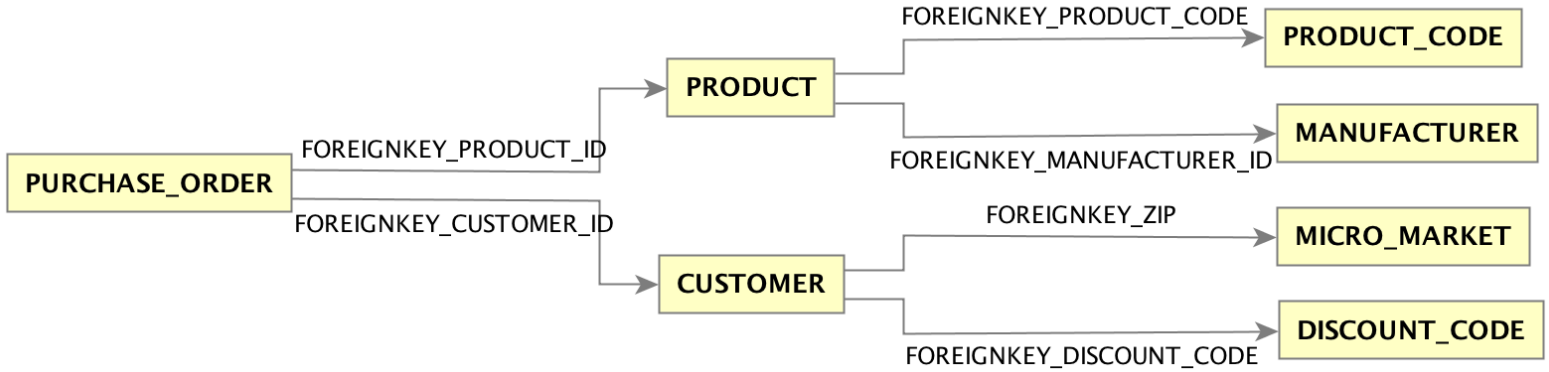


Vizualizácia „Sample Databázy“ (v DB vizualizéri)

- Rýchlo získate vizuálny prehľad o vzťahoch medzi tabuľkami



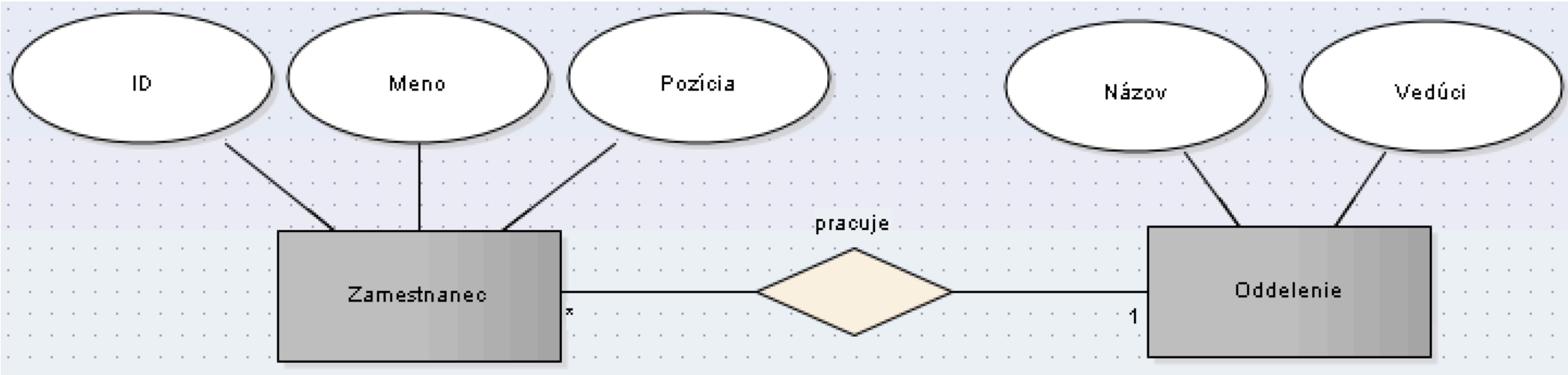
- Hierarchický pohľad (lepšie vidieť prítomné hierarchie):



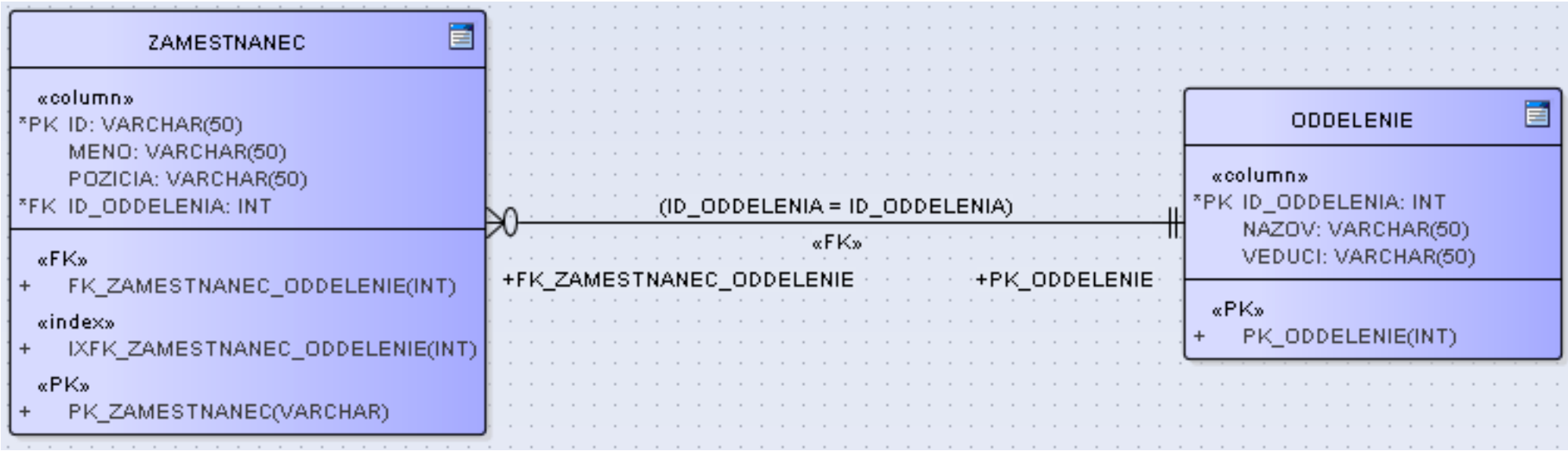
Všimnite si previazanie tabuliek pomocou cudzieho kľúča

Uvažujme príklad z predchádzajúcej prednášky

Koncepčný diagram:

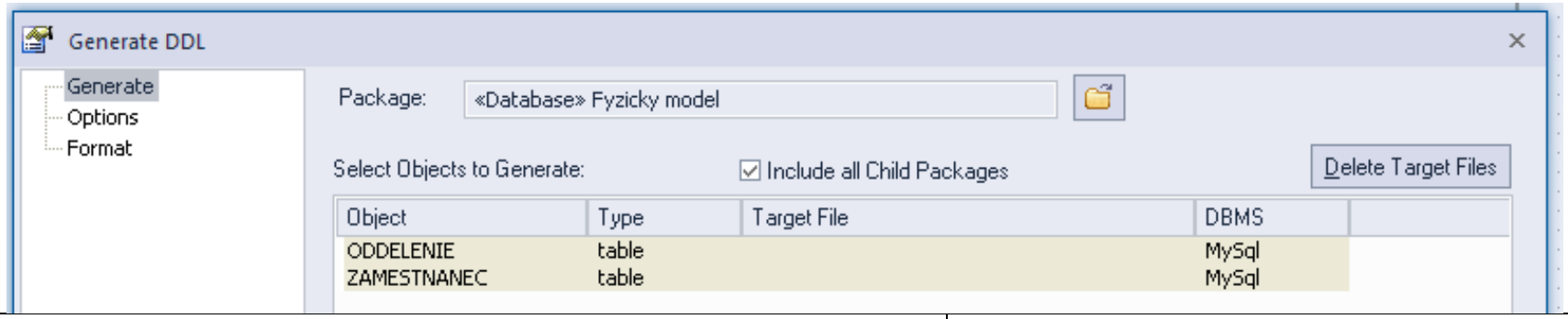


Fyzický model:



Vytvorenie štruktúry v databáze z DDL skriptov

1) V EA: zvoliť fyzický model → Code Engineering → Generate DDL
(alebo vytvoriť skripty manuálne)



2) Výsledok generovania DDL:

```
SET FOREIGN_KEY_CHECKS=0;
DROP TABLE IF EXISTS `ODDELENIE` CASCADE;
DROP TABLE IF EXISTS `ZAMESTNANEC` CASCADE;
CREATE TABLE `ODDELENIE`
(
        `ID_ODDELENIA` INT NOT NULL,
        `NAZOV` VARCHAR(50),
        `VEDUCI` VARCHAR(50),
        CONSTRAINT `PK_ODDELENIE` PRIMARY KEY (`ID_ODDELENIA`)
);
CREATE TABLE `ZAMESTNANEC`
(
        `ID` VARCHAR(50) NOT NULL,
        `MENO` VARCHAR(50),
        `POZICIA` VARCHAR(50),
        `ID_ODDELENIA` INT NOT NULL,
        CONSTRAINT `PK_ZAMESTNANEC` PRIMARY KEY (`ID`)
);
ALTER TABLE `ZAMESTNANEC`
 ADD INDEX `IXFK_ZAMESTNANEC_ODDELENIE` (`ID_ODDELENIA` ASC);
ALTER TABLE `ZAMESTNANEC`
 ADD CONSTRAINT `FK_ZAMESTNANEC_ODDELENIE`
        FOREIGN KEY (`ID_ODDELENIA`) REFERENCES `ODDELENIE` (`ID_ODDELENIA`)
            ON DELETE Restrict ON UPDATE Restrict;
SET FOREIGN_KEY_CHECKS=1;
```

3) Nakopírovať do SQL okna v netbeans
+ pridať na začiatok napr.:

```
DROP DATABASE IF EXISTS FIRMA;
CREATE DATABASE FIRMA;
USE FIRMA;
```

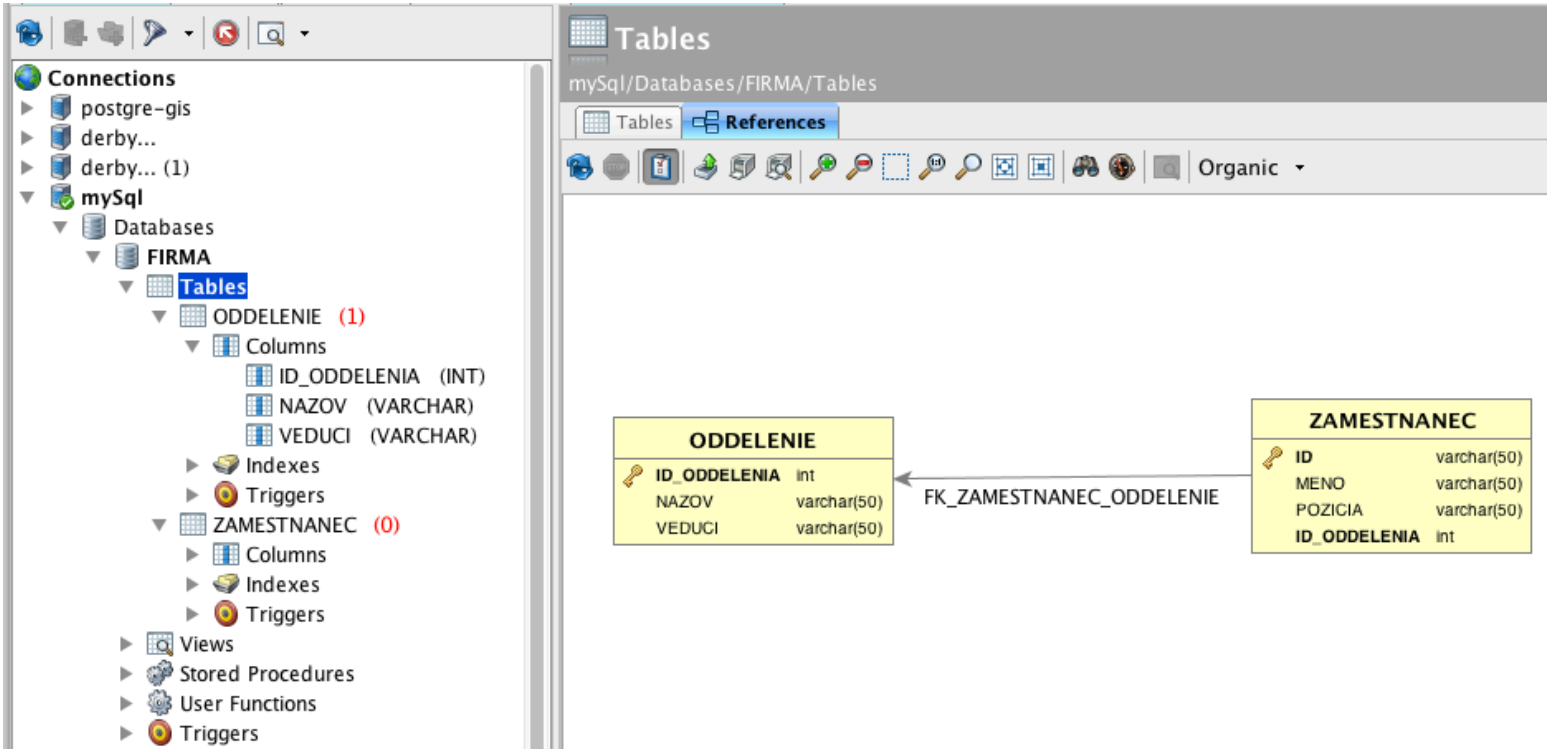
4) Alebo všetko uložiť do súboru
a použiť:

mysql –u uzivatel -p < subor_s_prikazmi

Poznámka: Preštudujte si syntax

1. príkazov CREATE TABLE ... vrátane
   1.1. NOT NULL
   1.2. CONSTRAINT
2. ALTER TABLE ...
3. SET ...

Výsledok spustenia DDL skriptov v MySQL database [DbVizualizer]



SQL základné príkazy

- vytvorenie databázy: „CREATE DATABASE ...“ (https://dev.mysql.com/doc/refman/5.7/en/create-database.html) + DROP DATABASE ...
- Vytvorenie tabuľky: „CREATE TABLE ...“ (https://dev.mysql.com/doc/refman/5.7/en/create-table.html) + DROP TABLE ...

Vytvorenie štruktúry v databáze

DDL príkazy v MySql:

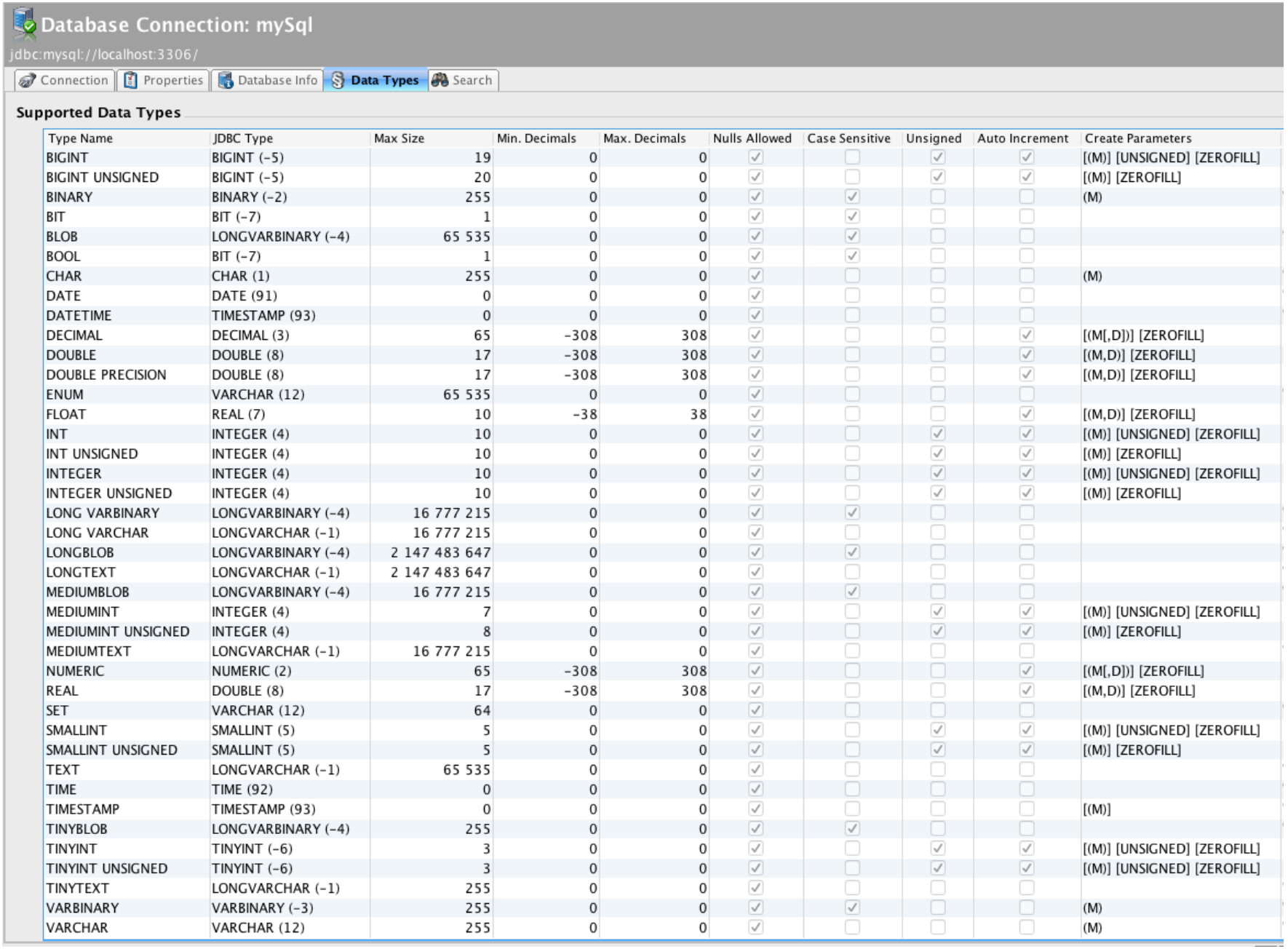
Objekt	**CREATE**	**DROP**	**ALTER**	**Iné**
DATABASE	X	X	X	
EVENT	X	X	X	
FUNCTION	X	X	X	
INDEX	X	X		
LOGFILE GROUP	X	X	X	
PROCEDURE	X	X	X	
SERVER	X	X		
TABLE	X	X	X	RENAME, TRUNCATE
TABLASPACE	X	X	X	
TRIGGER	X	X		
VIEW	X	X	X	

- napĺňanie tabuliek údajmi
  - INSERT INTO <tabuľka> (‚meno‘,‘dfsdf‘, ...)
- zmeny hodnôt
  - UPDATE <tabuľka> SET stlpec=vyraz, stlpec2=vyraz2 where stlpec5 like ‚kkk‘
- mazanie
  - DELETE FROM ...
- výber SELECT ....

Typy stĺpcov v MySQL

- Číselné (reálne čísla - NUMERIC, DECIMAL, DOUBLE, REAL, FLOAT; celé čísla TINYINT (1B) , SMALLINT (2B) , MEDIUMINT (3B), BIGINT (8B))
- Reťazcové a Textové typy
  - CHAR(N) – má práve N znakov, VARCHAR (N) – má max. N znakov (šetrí úložiskom), N=max 255
  - TEXT – pre viac ako 255 znakov, BLOB (Binary Large Object)
    - TINYTEXT/TYNIBLOB – 255 znakov/bajtov
    - TEXT/BLOB - 64KB znakov/bajtov
    - MEDIUMTEXT/MEDIUMBLOB - 16MB znakov/bajtov
    - LONGTEXT/LONGBLOG – 4GB znakov/bajtov
    - ENUM, SET
- Typy DATE a TIME
  - DATE, TIME, DATETIME, TIMESTAMP, YEAR,

Čo prezradí spojenie na DB



Database Connection: mySql

jdbc:mysql://localhost:3306/

Connection | Properties | Database Info | Data Types | Search

Supported Data Types

Type Name	JDBC Type	Max Size	Min. Decimals	Max. Decimals	Nulls Allowed	Case Sensitive	Unsigned	Auto Increment	Create Parameters
BIGINT	BIGINT (-5)	19	0	0	✓	☐	✓	✓	[(M)] [UNSIGNED] [ZEROFILL]
BIGINT UNSIGNED	BIGINT (-5)	20	0	0	✓	☐	✓	✓	[(M)] [ZEROFILL]
BINARY	BINARY (-2)	255	0	0	✓	✓	☐	☐	(M)
BIT	BIT (-7)	1	0	0	✓	✓	☐	☐	
BLOB	LONGVARBINARY (-4)	65 535	0	0	✓	✓	☐	☐	
BOOL	BIT (-7)	1	0	0	✓	✓	☐	☐	
CHAR	CHAR (1)	255	0	0	✓	☐	☐	☐	(M)
DATE	DATE (91)	0	0	0	✓	☐	☐	☐	
DATETIME	TIMESTAMP (93)	0	0	0	✓	☐	☐	☐	
DECIMAL	DECIMAL (3)	65	-308	308	✓	☐	☐	✓	[(M[,D])] [ZEROFILL]
DOUBLE	DOUBLE (8)	17	-308	308	✓	☐	☐	✓	[(M,D)] [ZEROFILL]
DOUBLE PRECISION	DOUBLE (8)	17	-308	308	✓	☐	☐	✓	[(M,D)] [ZEROFILL]
ENUM	VARCHAR (12)	65 535	0	0	✓	☐	☐	☐	
FLOAT	REAL (7)	10	-38	38	✓	☐	☐	✓	[(M,D)] [ZEROFILL]
INT	INTEGER (4)	10	0	0	✓	☐	✓	✓	[(M)] [UNSIGNED] [ZEROFILL]
INT UNSIGNED	INTEGER (4)	10	0	0	✓	☐	✓	✓	[(M)] [ZEROFILL]
INTEGER	INTEGER (4)	10	0	0	✓	☐	✓	✓	[(M)] [UNSIGNED] [ZEROFILL]
INTEGER UNSIGNED	INTEGER (4)	10	0	0	✓	☐	✓	✓	[(M)] [ZEROFILL]
LONG VARBINARY	LONGVARBINARY (-4)	16 777 215	0	0	✓	✓	☐	☐	
LONG VARCHAR	LONGVARCHAR (-1)	16 777 215	0	0	✓	☐	☐	☐	
LONGBLOB	LONGVARBINARY (-4)	2 147 483 647	0	0	✓	✓	☐	☐	
LONGTEXT	LONGVARCHAR (-1)	2 147 483 647	0	0	✓	☐	☐	☐	
MEDIUMBLOB	LONGVARBINARY (-4)	16 777 215	0	0	✓	✓	☐	☐	
MEDIUMINT	INTEGER (4)	7	0	0	✓	☐	✓	✓	[(M)] [UNSIGNED] [ZEROFILL]
MEDIUMINT UNSIGNED	INTEGER (4)	8	0	0	✓	☐	✓	✓	[(M)] [ZEROFILL]
MEDIUMTEXT	LONGVARCHAR (-1)	16 777 215	0	0	✓	☐	☐	☐	
NUMERIC	NUMERIC (2)	65	-308	308	✓	☐	☐	✓	[(M[,D])] [ZEROFILL]
REAL	DOUBLE (8)	17	-308	308	✓	☐	☐	✓	[(M,D)] [ZEROFILL]
SET	VARCHAR (12)	64	0	0	✓	☐	☐	☐	
SMALLINT	SMALLINT (5)	5	0	0	✓	☐	✓	✓	[(M)] [UNSIGNED] [ZEROFILL]
SMALLINT UNSIGNED	SMALLINT (5)	5	0	0	✓	☐	✓	✓	[(M)] [ZEROFILL]
TEXT	LONGVARCHAR (-1)	65 535	0	0	✓	☐	☐	☐	
TIME	TIME (92)	0	0	0	✓	☐	☐	☐	
TIMESTAMP	TIMESTAMP (93)	0	0	0	✓	☐	☐	☐	[(M)]
TINYBLOB	LONGVARBINARY (-4)	255	0	0	✓	✓	☐	☐	
TINYINT	TINYINT (-6)	3	0	0	✓	☐	✓	✓	[(M)] [UNSIGNED] [ZEROFILL]
TINYINT UNSIGNED	TINYINT (-6)	3	0	0	✓	☐	✓	✓	[(M)] [UNSIGNED] [ZEROFILL]
TINYTEXT	LONGVARCHAR (-1)	255	0	0	✓	☐	☐	☐	
VARBINARY	VARBINARY (-3)	255	0	0	✓	✓	☐	☐	(M)
VARCHAR	VARCHAR (12)	255	0	0	✓	☐	☐	☐	(M)

Čo je index a na čo je dobrý

- V SRDB sú na diskoch uložené informácie 4 kategórií
  - DATA – vlastný obsah databázy
    - Uložené sú v indexovaných sekvenčných súboroch
  - Metadáta v slovníku údajov – popis schémy DB a integritných obmedzení
  - Štatistiky- informácie o uložených údajoch
  - **Indexy – podpora efektívneho prístupu k údajom**
    - Indexové súbory obsahujú pomocné informácie ako rýchle pristupovať k datovým suborom
    - Indexové súbory sú podstatne menšie ako datové
    - **Fungujú na princípe rýchleho nájdenia dvojice (vyhľadávací kľúč, fyzická adresa záznamu)**
- Sledované kritériá pre indexy
  - **Urýchlenie** vyhľadávania pri použití indexu
  - **Spomalenie** pri vkladaní a mazaní údajov
- Index sa automaticky robí pre primárny kľúč
  - index sa môže vytvoriť aj nad inými stĺpcami, na záznamy zvyčajne ukazuje sprostredkovane pomocou primárneho kľúča
- v SQL vytvára sa pomocou „CREATE INDEX ...“
- Fyzická forma indexu
  - ISAM (indexed sequential access method) - pre málo sa meniace štruktúry, použitie hash máp
  - B^+ stromy - pre dynamicky sa meniace štruktúry
    - stromy je dôležité udržiavať vyvážené (všetky cesty od koreňa do ľubovoľného listu sú rovnako dlhé), lebo počet prístupov na disk pri vyhľadávaní odpovedá dĺžke stromu

Poznámka k indexom

- index môže, ale nemusí vyžadovať jedinečnosť (UNIQUE)
- pre typy CAHR a VARCHAR môžeme indexáciu
  - obmedziť na niekoľko prvých znakov: create index mojIndex on ZAMESTNANCI (pozicia(5))
  - alebo naopak rozšíriť index na FULLTEXT (môže vzniknúť viacero záznamov v indexe pre jeden kľúč, napr. pri zázname „pes a macka jedli“ vzniknú 4 fulltextové záznamy)
- indexy na textových typoch nie sú také účinne ako na numerických!
- Ak nie je k dispozícii index, robí sa zakaždým „FULL TABLE SCAN“

Hashmapa (HM)

- nazýva sa aj hešovacia tabuľka je dátová štruktúra, pomocou ktorej sa implementuje asociatívne pole. t.j. štruktúra, ktorá mapuje kľúče na HODNOTY
- hešovacia tabuľka používa hešovaciu funkciu na vypočítanie INDEXu do poľa, kde sa dá nájsť HODNOTA:

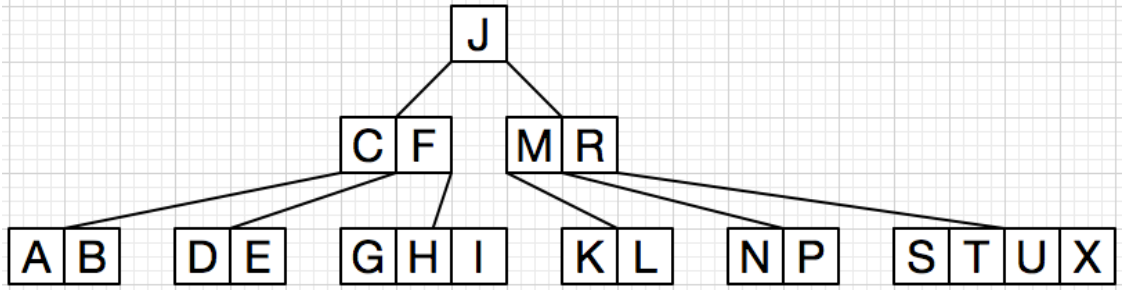
Index=hashFnc(KLUC, VELKOST_POLA)	Index=hashFnc(KLUC)%VELKOST_POLA

- kolízia = rôzne kľúče vedú k tomu istému indexu
  - kolíziám sa nedá vyhnúť, pravdepodobnosť, že aspoň dva indexy sú v kolízii je minimálne $p(pk, vp) \approx 1 - e^{-pk(pk-1)/2vp}$, kde pk=počet kľúčov, vp=veľkosť poľa, t.j. ak máme napr. 1000 kľúčov a veľkosť poľa 1milión, potom p=0,3932, t.j. s cca 39% pravdepodobnosťou máme kolíziu
  - minimalizácia počtu kolízií → zvolením vhodnej hešovacej funkcie
  - riešenie kolízie → napr. triviálne je, že v každej HODNOTE je zoznam, z ktorého sa volí, zvyčajne má 0-1 záznamov, pri dobrej hešovacej funkcii má zriedkavo viac ako pár hodnôt

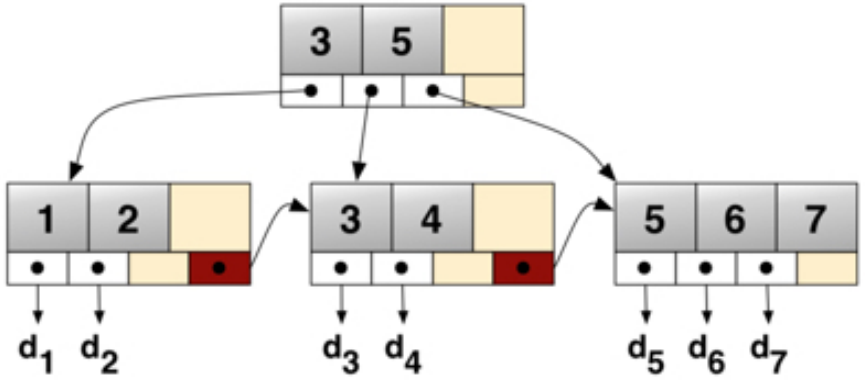
B strom, B+ strom

- B strom je samovyvažujúca sa stromová údajová štruktúra, ktorá udržiava údaje zotriedené a dovoľuje vyhľadávania, vkladania, mazania ... všetko v **logaritmickom** čase.
  - *m*-árny strom, má interné uzly a listy (uzly na najnižšej úrovni)
  - v každom uzle sú KV (key/value) páry
- B+ strom je rozšírenie B stromu, kde
  - KV páry sú uložené len v listoch, kópie kľúčov uložené aj v interných uzloch
  - listy môžu byť navzájom prelinkované (kvoli sekvenčnému prístupu)

Príklad B stromu pre m=5 a hodnoty {A G F B K D H M J E S I R X C L N T U P}:

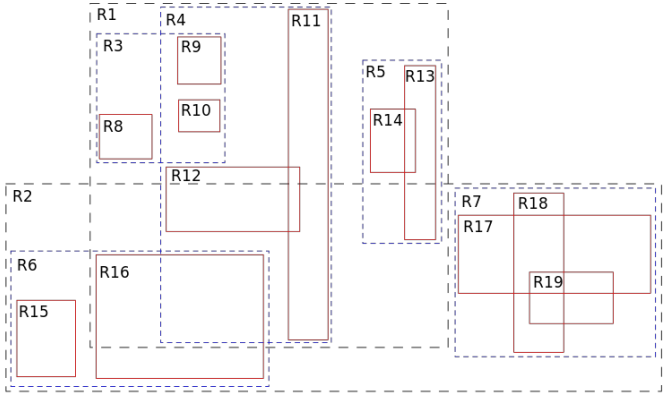


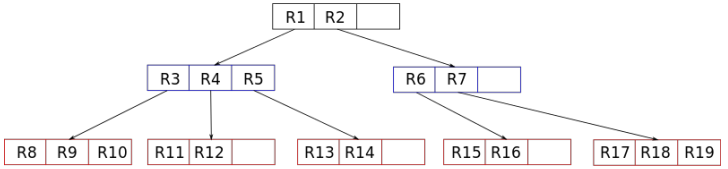
Príklad B+ stromu pre m=4:



Základné druhy indexov

- **Klastrovaný/neklastrovaný** index:
  - pri neklastrovanom indexe sú kľúče zotriedené (v strome, hashmape), ukazujú na nezotriedené záznamy – záznamy môžu byť uložené v inom poradí.
  - pri klastrovanom indexe sú záznamy uložené v tom istom poradí ako kľúče
    - Veľmi rýchly sekvenčný prístup k záznamom
    - Len jeden klastrovaný index môže byť vytvorený nad jednou tabuľkou.
- **Bitmapový** ukladá data ako bitmapy - efektívny v prípadoch, keď sa hodnoty atribútu často opakujú. Napr. atribút pohlavie osob. Vtedy je tento index oveľa rýchlejší ako „stromové“ (B, B+) indexy
- **Hustý** (dense) index: súbor KV párov pre každý záznam, V je pointer na záznam do utriedeného dátového súboru.
- **Riedky** (sparse) index: súbor KV párov pre každý blok záznamov, V je pointer na blok do utriedeného dátového súboru
- **Otočený** index: Hodnoty kľúčov sa otáčajú pre uložením. Napr. ak kľúč sú monotónne rastúce čísla (123, 124, 125, ...) tieto budú (321, 421, 521, ...) – B strom sa zaplní rovnomernejšie.
- **Priestorový** index (v priestorových databázach) – napr. pomocou R-stromov (viď obrázok vpravo).
- ...

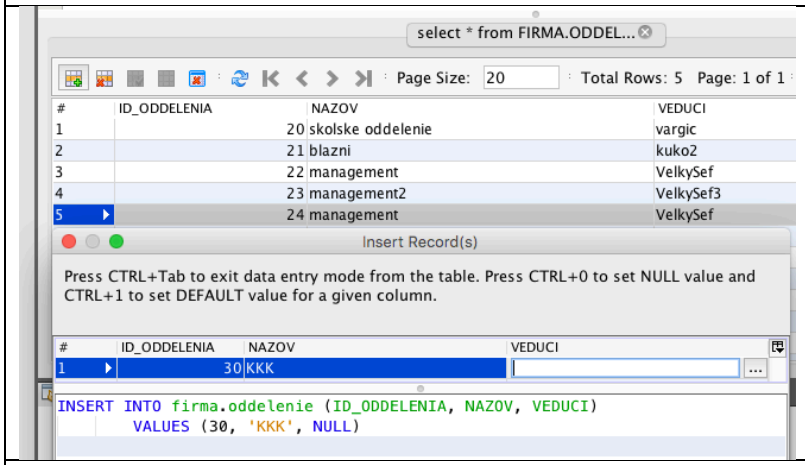


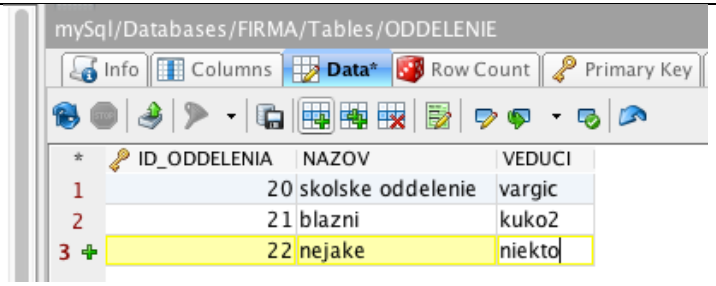


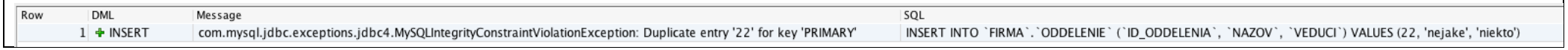
Vkladanie dát

- Pomocou SQL (z GUI, suboru, ...) pomocou INSERT/REPLACE (replace pri kolízii aj nahrádza)
  - Napr.: insert into FIRMA.ODDELENIE(ID_ODDELENIA, NAZOV, VEDUCI) VALUES (22, 'management','VelkySef');
  - Kontrola: select * from FIRMA.ODDELENIE;
- Pomocou GUI:

Netbeans	DbVizualizer







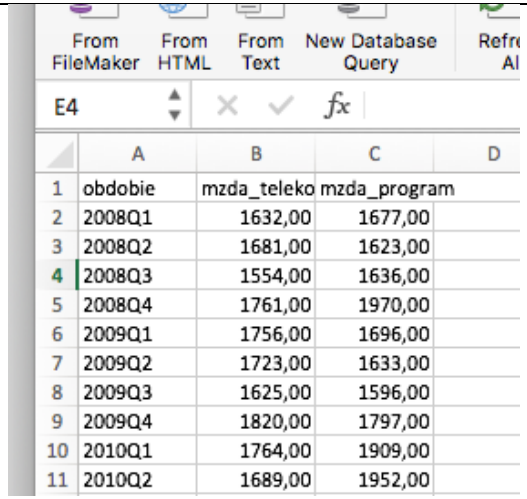
- Import údajov
  - napr zoberme si štatistický úrad (http://www.statistics.sk/..., napr databázu Slovstat
  - Sú tam údaje o platoch v sektore telekomunikácií, informatike atď ... ☺
    - excel export → Dbvizualizer import do pripravenej tabuľky (wizzard na mapovanie polí ..)
  - MySql: LOAD XML ...
    - môžeme údaje v XML aj exportovať:

```
/usr/local/mysql/bin/mysql -u root -p --xml -e 'SELECT * FROM
FIRMA.slovstat_platy' > platy.xml
```

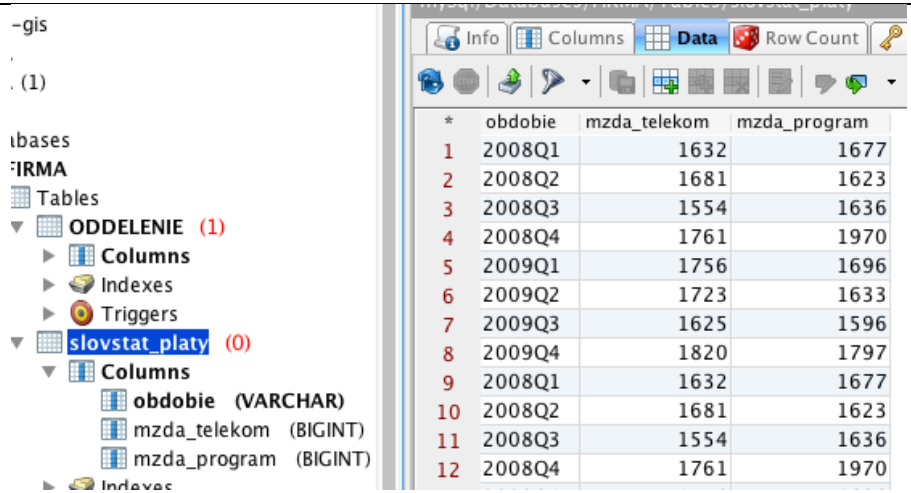
  - MySql: LOAD DATA (txt ...)
- naprogramovať si nejaký nástroj na import, napr. v jave ?

Slovstat export →

Excel úprava (transpose)



DbVizualiser (Import Table Data)



Exportované XML (platy.xml)

```xml
<?xml version="1.0"?>

<resultset statement="SELECT * FROM FIRMA.slovstat_platy
" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="obdobie">2008Q1</field>
    <field name="mzda_telekom">1632</field>
    <field name="mzda_program">1677</field>
  </row>

  <row>
    <field name="obdobie">2008Q2</field>
    <field name="mzda_telekom">1681</field>
    <field name="mzda_program">1623</field>
  </row>
```

...

Podporované 3 rôzne XML formy pri importe (LOAD XML)

1
```xml
<row column1="value1" column2="value2" .../>
```

2
```xml
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

3
```xml
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

Text export

```
SELECT * INTO OUTFILE 'data.txt'
  FIELDS TERMINATED BY ','
FROM TABLE FIRMA.slovstat_platy
```

Vystup:

```
2008Q1,1632,1677
2008Q2,1681,1623
...
```

Text import

```
LOAD DATA INFILE 'data.txt' INTO TABLE
FIRMA.slovstat_platy FIELDS TERMINATED BY ',';
```

Aktualizácia a odstraňovanie údajov

- UPDATE – slúži na aktualizáciu obsahu riadkov
  - UPDATE tabulka SET atribut=hodnota [WHERE nejakaPodmienka] [...]
- DELETE – mazanie tabuliek, alebo ich častí ...
  - DELETE FROM tabulky USING tabulky [WHERE nejakaPodmienka]
    - klauzula USING sa používa vtedy, ak množina tabuliek použitá vo WHERE je väčšia
- TRUNCATE – vyprázdnenie celej tabuľky

Ako použiť DELETE ak máme v tabuľke cudzí kľúč? Dajme tomu, že máme údaje:

```
mysql> SELECT * FROM FIRMA.ZAMESTNANEC;
+-----+------+-------------+--------------+
| ID  | MENO | POZICIA     | ID_ODDELENIA |
+-----+------+-------------+--------------+
| 100 | Rado | zamestnanec |           20 |
| 101 | Fero | upratovac   |           21 |
+-----+------+-------------+--------------+
```

```
mysql> SELECT * FROM FIRMA.ODDELENIE;
+--------------+--------------------+-----------+
| ID_ODDELENIA | NAZOV              | VEDUCI    |
+--------------+--------------------+-----------+
|           20 | skolske oddelenie  | vargic    |
|           21 | blazni             | kuko2     |
|           22 | management         | VelkySef  |
```

- chcem zmazať oddelenie 20:
  - `delete from FIRMA.ODDELENIE where ID_ODDELENIA=20;`
- výsledok
  - `Error code 1451, SQL state 23000: Cannot delete or update a parent row: a foreign key constraint fails`
- najprv treba zmazat zamestnancov pri ktorých cudzí kľúč na dané oddelenie ukazuje a potom samotné oddelenie
  - `delete from FIRMA.ZAMESTNANEC where ZAMESTNANEC.ID_ODDELENIA=20;`
  - `delete from FIRMA.ODDELENIE where ODDELENIE.ID_ODDELENIA=20;`

Druhy SQL príkazov

- definícia dát (CREATE, DROP, ALTER, RENAME, TRUNCATE)
- manipulácia s datami (CALL, DELETE, DO, HANDLER, INSERT, LOAD, REPLACE, SELECT, UPDATE)
- tranzakcie a uzamkýnanie (START TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT, LOCK TABLES, UNLOCK TABLES, SER TRANSACTION)
- replikácia (SHOW BINARY LOGS, ...)
- prepared (PREPARE, EXECUTE DEALLOCATE PREPARE)
- compoud (BEGIN...END, DECLARE, ...)
    - flow control (CASE, IF, ITERATE, LEAVE, LOOP, REPEAT, RETURN, WHILE)
    - kurzory (CLOSE, DECLARE, FETCH, OPEN)
    - ...
- administrácia
    - užívatelia (CREATE USER, DROP USER, ...)
    - tabuľky (ANALYZE TABLE, ...)
    - funkcie (STREATE FUNCTION, ...)
    - SHOW AUTHORS ...
    - ...
- utility (DESCRIBE, EXPLAIN, HELP, USE)

SELECT a jeho syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

poďme na vysvetlenie pomocou prikladov **...**

Príklady použitia SELECTu

SELECT * FROM databaza.tabulka;	Vybere všetky riadky tabuľky v danej databáze, všetky stĺpce
SELECT * FROM tabulka;	Vybere všetky riadky tabuľky, všetky stĺpce
SELECT s1, s2 FROM table;	Vybere všetky riadky tabuľky, stĺpce s1, s2
SELECT s1 AS mojAlias from tabuľka;	Vybere všetky riadky tabuľky, stĺpec s1 a nazve ho mojAlias
SELECT * FROM tabulka WHERE podmienka;	Vybere riadky tabuľky, pre ktoré je splnená podmienka, všetky stĺpce

WHERE podmienka môže byť napr.:

- operator rovnosti, napr. stĺpec='hodnota'
- operátor nerovnosti !=, <>, >, <, >=, <=, môže byť kombinovaný aj s aritmetickými operáciami
- IS NULL
- Logické operátory AND, OR, NOT, &&, ||, XOR, kombinácie logických výrazov pomocou „(„ a „)“
- n BETWEEN min AND max (podmienka na rozsah hodnôt)
- n IN množina – test, či n patrí do množiny prvkov, napr. „ (jablko, hruska, slivka)“
- n > ALL(množina) – pravda ak n > ako všetky prvky množiny
    - de facto sa hodnota porovnáva zo všetkými prvkami jednotlivo a medzi porovnaniami je AND
    - ako operátor sa dá použiť =, !=, <>, >, <, >=, <=,
- n > ANY(množina), resp. n>SOME (množina)
    - de facto sa hodnota porovnáva sa zo všetkými prvkami jednotlivo a medzi porovnaniami je OR
    - ako operátor sa dá použiť =, !=, <>, >, <, >=, <=,

Napr:

- select * from CUSTOMER where DISCOUNT_CODE = 'L' and customer_id>200
- select * from CUSTOMER where discount_code in ('L','M')
- select * from CUSTOMER where customer_id = ANY(select customer_id from PURCHASE_ORDER where quantity>100)

Príklady použitia SELECTu 2

(pozor v DERBY nefungujú všetky príklady)

SELECT count(*) FROM tabulka ;	Vráti počet vyhovujúcich záznamov
SELECT DISTINCT s1 FROM tabulka;	Vráti rozdielne hodnoty
SELECT count (DISTINCT s1) FROM tabulka;	Vráti počet rozdielnych hodnôt
SELECT count(*), s1 FROM tabulka GROUP by s1;	Vráti počty výskytov aj hodnoty pre jednotlivé hodnoty stĺpca s1
SELECT count(*), s1 FROM tabulka GROUP by s1 HAVING podmienka;	Podmienka HAVING sa vzťahuje na vytvorené skupiny (nie na jednotlivé záznamy ako WHERE)
SELECT * FROM tabulka ORDER BY s1 ASC	Vybere všetky riadky tabuľky, všetky stĺpce, zotriedi ich vzostupne podľa stĺpca s1
SELECT * FROM tabulka limit [N,] M	Vybere prvých M riadkov (prípadne s offsetom N), všetky stĺpce

Príklady:

- select CUSTOMER_ID, count(*) as ORDER_CNT, from PURCHASE_ORDER group by CUSTOMER_ID
- select CUSTOMER_ID, count(*) as ORDER_CNT from PURCHASE_ORDER group by CUSTOMER_ID **order by ORDER_CNT desc**
- select CUSTOMER_ID, count(*) as ORDER_CNT from PURCHASE_ORDER group by CUSTOMER_ID **having CUSTOMER_ID > 10** order by ORDER_CNT desc

Príklady použitia SELECTu 3

(pozor v DERBY nefungujú všetky príklady)

SELECT N+M;	Vráti súčet N+M
SELECT aritmetický_výraz;	Vráti hodnotu ľubovoľného aritmetického výrazu, napr. SIN(1+SQRT(10)), podporované sú napr. abs, ceil, floor, mod, div, power, rand, round, sqrt, sin, exp, ln, log, sign, …
SELECT * from tab1 where s1= binary 'textstring';	Pri porovnávaní berie do úvahy aj veľkosť písmen (defaultne to SQL nerobí)
SELECT s1, IF(podm, val1, val2) FROM tabulka;	Okrem hodnot s1 vracia aj hodnoty ná základe výsledku IF, ak je podmienka splnená vracia val1, inak val2
SELECT s1, CASE WHEN podm THEN val1 WHEN podm THEN val2 ELSE val3 FROM tabulka;	Fungovanie ako v SWITCHi, ak aktuálna podmienka nevyhovuje, prechádza sa na ďalšiu a ďalšiu
SELECT CONCAT (s1, s2) AS full_name FROM tabulka	Vráti pospájané reťazce hodnôt v stĺpcoch s1 a s2 a nazve to celé full_name
SELECT * FROM tabulka WHERE s1 LIKE 'searchstring';	Volí na základe textového porovnania, znak % v reťazci nahrádza ľubovoľný počet znakov, napr. '%auto%' hľadá autá …

Ďalšie vyhľadávacie funkcie ako LIKE:

- STRCMP – syntax ako jazyku C
- RLIKE – porovnávanie regulárnych výrazov
- MATCH – fulltext vyhľadávanie

Príklad:

- SELECT * from CUSTOMER where ZIP like '95%';