Týždeň 4 – 5 - 6

A) Dátové modely, schémy a inštancie, integrita dát - 1
B) Návrh relačných databáz a ER diagramy
C) Vytvorenie databáz, tabuliek a integrita dát - 2

A) Dátové modely, schémy a inštancie, integrita dát - 1

1) Dátové modely
2) Schémy, inštancie a stav databázy
3) Princípy/pravidlá relačných databáz
4) Integrita dát a obmedzenia - 1

B) Návrh relačných databáz a ER diagramy

1) ER diagramy
2) Vzťahy
3) Slabé entity
4) Rozhodovania pri modelovaní

C) Vytvorenie databáz, tabuliek a integrita dát – 2

1) Integrita dát - 2
  a) Narušenie a zabezpečenie integrity dát
  b) Obmedzenia
2) Primárny a sekundárny kľúč - PK a FK, UNIQUE
  Vytvorenie a odstránenie PK a FK, NOT NULL
  a) V rámci definície tabuľky
  b) pomocou ALTER TABLE
    - bez CONSTRAINT => generovane meno
    - pomocou CONSTRAINT s menom
  c) UNIQUE
  d) Kompozitný kľúč
  e) DEFAULT hodnota
3) CHECK constraint, triggery
4) Spätná kontrola pri **ALTER TABLE**
5) … ADD / DROP COLUMN
6) Zistenie obmedzení programovo
7) ENUM obmedzenie
8) UPDATE a DELETE
9) CREATE TABLE ... SELECT / LIKE
10) Indexy

Sémantika – význam; Integrita – celistvosť bez nedostatku, poškodenia;
Valid – platný; a Konzistentnosť – dôslednosť bez protirečení, anomálií;

A) Dátové modely, schémy a inštancie, princípy relačných DB, integrita dát - 1

1) Dátové modely
2) Schémy, inštancie a stav databázy
3) Princípy/pravidlá relačných databáz
4) Integrita dát a obmedzenia - 1

1) Dátové modely

Dátový model

- Je **popis** štruktúry dát, ich typov, ohraničení a vzťahov poskytnutím definície a formátu dát
- poskytuje nutné nástroje na dosiahnutie dátovej abstrakcie – ako sa symboly a koncepty vzťahujú k reálnemu svetu, skrývajúc štruktúru a parametre fyzickej pamäte
- väčšina dátových modelov obsahuje aj množinu základných operácii pre špecifikáciu dopytov a aktualizáciu databázy.

Tri typy dátových modelov, konceptuálne, logické a fyzické dátové modely sú fázy návrhu úspešnej databázovej aplikácie, využívajúc rôzne stupne abstrakcie. Konceptuálne a logické dátové modely využívajú pojmy ako entita, atribút a vzťah:

- entita reprezentuje **koncept** alebo [všeobecný] objekt z reálneho sveta, ako napríklad zamestnanec alebo projekt, ktorý je opísaný v databáze; rozlišujú sa tu tri pojmy, ktoré treba vždy jasne definovať (pozri nižšie): *entita, entitný typ a inštancia entity*
- atribút reprezentuje vlastnosť, ktorá bližšie popisuje entitu, ako napríklad meno zamestnanca alebo jeho mzda
- vzťah medzi dvomi alebo viacerými entitami reprezentuje asociáciu medzi dvomi alebo viacerými entitami, ako napríklad vzťah zamestnanca k projektu, na ktorom zamestnanec pracuje
- doména – kombinácia typu a ohraničenia, ktoré definuje množinu možných hodnôt atribútu.
-
- Konceptuálny DM – je séria tvrdení o povahe DB (napr. organizácie), ktorých **význam** je opísaný názvami entít a ich vzťahmi, ktoré sú založené na požiadavkách a analýze. Mali by byť **nezávislé** (ER, UML) od konkrétnej technológie manažmentu dát.

- Logický DM – obohacuje konceptuálny DM o informácie o doméne a dátových štruktúrach (relational, object, XML, graph). Zahŕňa atribúty pre všetky entity, primárne a cudzie kľúče. Na tomto stupni sa robí normalizácia.
- Fyzický DM – obsahuje všetky tabuľkové štruktúry, vrátane názvov stĺpcov, ich dátových typov a ohraničení. Môžeme ho považovať za mapovanie logického DM na databázový model daného DBMS (Database Management System).

	konceptuálny	logický	fyzický
Názvy entít	✓	✓	
Vzťahy medzi entitami	✓	✓	
Atribúty		✓	
Normalizácia		✓	
Primárne kľúče		✓	✓
Cudzie kľúče		✓	✓
Názvy tabuliek		✓	✓
Názvy stĺpcov			✓
Dátové typy stĺpcov			✓

Dátový model je:

1. Reprezentácia, popis dát – štruktúra, vzťahy napr. relačný model – tabuľky, alebo pološtruktúrované modely – stromy, XML	DDL
2. Ohraničenia – typy	
3. Operácie nad dátami	DML

2) Schémy, inštancie a stav databázy

V každom dátovom modeli je dôležité rozlišovať medzi

- popisom databázy a
- databázou samotnou.

Popis databázy sa realizuje databázovou schémou a obsahuje schémy tabuliek (entít), ktoré sa špecifikujú počas návrhu databázy a neočakáva sa, že by sa mali často meniť.

Zobrazenie schémy sa nazýva diagram schémy (buď DB alebo entity).
Dáta, ktoré sú v databáze v danom momente, sa nazývajú stav databázy alebo inštancia schémy (resp. entity).

Pripomíname, že namiesto dvojice entita a inštancia entity sa niekedy používa dvojica entitný typ a entita!

DBMS má uložené popisy konštrukcií schém a ohraničenia ako metadáta. DBMS je čiastočne zodpovedný za to, aby každý stav databázy bol platný stav – teda taký stav, v ktorom je dodržaná štruktúra a **ohraničenia** špecifikované v schéme.

Relačný model
Základná **konštrukcia** reprezentácie dát v relačnom modeli je relačná schéma:

$R(A_1:D_1, \ldots, A_n:D_n)$:

- názov relácie R
- názvy atribútov A_i
- domény atribútov D_i

Relačná schéma korešponduje s entitou.

Inštancie schémy:

- relácia, ako množina n-tíc (DB teória)
- tabuľka so stĺpcami a riadkami (SQL)

3) Princípy/pravidlá relačných databáz

Codd definoval dvanásť princípov pre relačné databázy:
„1. Informácie sú logicky prezentované v tabuľkách.

2. Údaje musia byť logicky prístupné podľa: tabuľky, primárneho kľúča a stĺpca.

3. S nulovými hodnotami sa musí narábať jednotne ako s „chýbajúcimi informáciami“, nie ako s prázdnymi reťazcami, medzerami alebo nulami.

4. Metadáta musia byť v databáze uložené rovnako ako bežné údaje.

5. Jeden jazyk musí byť schopný definovať: údaje, zobrazenia, obmedzenia integrity, autorizáciu, transakcie a manipuláciu s údajmi.

6. Pohľady musia zobrazovať aktualizácie ich základných tabuliek a naopak.

7. Jedna operácia musí byť schopná načítať, vložiť, aktualizovať alebo vymazať údaje.

8. Dávkové a konečné operácie sú logicky oddelené od fyzického ukladania a prístupových metód.

9. Dávkové a konečné operácie môžu zmeniť databázovú schému bez toho, aby ste ju museli znovu vytvárať.

10. Obmedzenia integrity musia byť dostupné a uložené v metadátach RDB, nie v aplikačnom programe.

11. Jazyk manipulácie s údajmi v relačnom systéme by sa nemal starať o to, kde alebo ako sú fyzické údaje distribuované, a nemal by vyžadovať zmenu, ak sú fyzické údaje centralizované alebo distribuované.

12. Akékoľvek spracovanie riadkov vykonané v systéme musí vyhovovať rovnakým pravidlám integrity a obmedzení, ako operácie spracovania súborov."

4) Integrita dát a obmedzenia - 1

Integrita dát je jeden z najdôležitejších aspektov návrhu databázy. Tri typy integrity sú:

A) integrita entít (horizontálna) ⇔ integrita [*inštancií*] entít
B) integrita domén (vertikálna) a
C) referenčná integrita (krížová).

Integrita entít a referenčná integrita tvoria dve zásadné pravidlá pre relačný model.

Integrita dát sa vzťahuje na platnosť dát, teda či sú dáta v databáze *korektné* v súlade s definovanými obmedzeniami **bez nedostatku** a logicky *konzistentné* **bez protirečení**. Databáza by mala obsahovať iba *platné* hodnoty (prípustné, správne – *valid*, accurate, correct). Napríklad

a) vkladanie riadku do tabuľky, ktorý už existujúce
b) ak stĺpec môže obsahovať iba kladné celé čísla, hodnota -1 nie je povolená
c) stĺpec, ktorý sa odvoláva na *iný* stĺpec druhej tabuľky, nemá obsahovať hodnoty, ktoré nie sú v tom *inom* stĺpci (primárny kľúč).

Kontrola platnosti dát zahŕňa: kontrolu typu dát, rozsahu dát, logickú konzistenciu (dátum2 >= dátum1), ...

Integrity a obmedzenia

Obmedzenia integrity dát

- sú pravidlá pre dáta, ktoré zodpovedajú požiadavkám
- predstavujú sémantickú správnosť.

Teda obmedzenia integrity dát sú podmienky, ktoré by dáta definované v relačnej schéme mali spĺňať – databáza môže obsahovať iba takéto dáta. Na zabezpečovanie troch typov integrity dát sa používajú rôzne obmedzenia:

A) **integrita entít** (riadok) - primárny kľúč alebo unikátne obmedzenie (UNIQUE) *zabezpečujú*, že databáza je bez duplicity dát a neobsahuje žiadne ekvivalentné riadky
   - *NOT NULL* obmedzenie - stĺpce označené ako NOT NULL nemajú obsahovať hodnoty NULL

B) **integrita domén** (stĺpec) - obmedzenie domény *zabezpečuje* platnosť a logickosť dát v stĺpcoch [varchar(n), int vs decimal]

C) **referenčná integrita** (vzťah) - obmedzenie cudzím kľúčom *zabezpečuje* spoľahlivosť a platnosť vzťahov v databáze. Vzťah medzi primárnym kľúčom základnej tabuľky a cudzím kľúčom inej tabuľky musí byť zachovaný – napríklad, primárny kľúč, na ktorý sa cudzí kľúč odkazuje, nemôže byť zmazaný.

   S referenčnou integritou súvisí kardinálne a účastnícke obmedzenie (pozri ER diagramy)
   - kardinálne obmedzenie entít vo vzťahu definuje, koľko inštancií jednej entity súvisí s inštanciami inej entity. Vyjadruje **existenciu** jedného typu entity v relácii k druhému typu entity, tzn. prípustné maximum (1,…N) **počtu** entít vo vzťahu.
     Nech A, B sú dve entity. Tri základné vzťahy sú:
     - 1:1 – inštancia entity A je viazaná na najviac jednu inštanciu entity B a naopak
     - 1:N - inštancia entity typu A môže byť viazaná na viac ako jednu inštanciu entity typu B, ale nie naopak
     - M:N - viac inštancií entity A môže byť viazaných na jednu inštanciu entity B a naopak, viac inštancií entity B môž byť viazaných na jednu inštanciu entity A
   - účastnícke obmedzenie entity vo vzťahu vyjadruje **nutnosť** jedného typu entity v relácii a existenciu druhého typu, tzn. možného minima (0, 1) počtu entít vo vzťahu
     - 0 indikuje neúplnú účasť (môžeme mať niekoľko entít, ktoré sa nepodieľajú na vzťahu)
     - 1 indikuje úplnú účasť (každá entita sa musí podieľať na vzťahu)

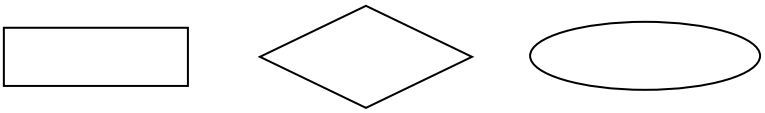
Reláčné databázové **aplikácie** na pozadí sa opierajú o RDB a v prezentačnej vrstve o OOP jazyk (C#, Java) či JavaScript. Na preklenutie priepasti medzi RDB a OOP je možné používať rôzne riešenia, ako ORDB, ORM alebo SP (písané v jazyku ako Microsoft .NET for SQL Server, Java for Oracle). Ch.J. Date navrhuje iné riešenie, valid D (data language specification), ktorý má vlastnosti odlišné od RDB.

B) Návrh relačných databáz a ER diagramy

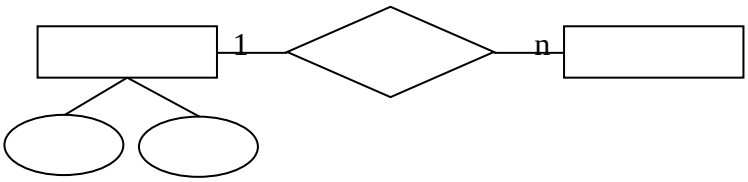
1) Entity a ER diagramy
2) Vzťahy
3) Slabé entity
4) Rozhodovania pri modelovaní
a) Reverse engineering in MySQL
c) VZŤAH 1:1
d) VZŤAH m:n
5) Príklad

1) Entity a ER diagramy

Prvým krokom pri návrhu RDB je získanie požiadaviek, po ktorom nasleduje modelovanie. Modelovať môžeme pomocou **ER diagramov** (**Entitno-relačné diagramy,** ER - Entity Relation), ktoré používajú znaky

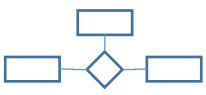


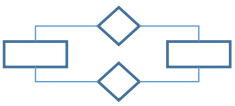
entita vzťah atribút



ER model interpretuje DB/úlohu ako množinu entít so zodpovedajúcimi atribútmi a vzťahmi medzi nimi.

- **Entita**/entytný typ je vecný/všeobecný objekt, osoba, miesto alebo udalosť (ich množina), ktorú je možné jednoznačne identifikovať v rámci úlohy.
- **Atribút** - množina atribútov určuje, charakterizuje objekt.
  Kľúč (kľúčový atribút) je súčasťou jedinečnej identifikácie.
  Arita je počet atribútov.
- **Vzťah** - binárny





  - rekurzívny
  - ternárny: 3 entity: prednášajúci, predmet, študent; 1 atribút: vyučuje
  - viacnásobný: 2 entity: oddelenie, personál; 2 vzťahy: je člen, je vedúci

Dva typy obmedzení vzťahu sú: kardinalita a účasť/členstvo (pozri nižšie).

Po získaní požiadaviek modelovanie znamená:
- analyzovať úlohu
  - identifikovať **entity**

- identifikovať **vzťahy** a určiť, či sú typu 1:1, 1:n alebo m:n.
- určiť domény a **atribúty** pre entity a vzťahy
- určiť primárne kľúče pre množinu entít

- načrtnúť **ERD** (entity-relationship diagram)
- kontrolovať, či ERD zodpovedá požiadavkám úlohy
(- transformácia ERD do databázového modelu pomocou DBMS)

2) Vzťahy

Pri modelovaní relačných DB dôležitú úlohu zohráva početnostná charakterizácia vzťahu, ktorá vyjadruje, že jednému riadku tabuľky koľko partnerských riadkov inej tabuľky zodpovedá. V ER diagramoch **početnosť** môžeme popísať kardinalitou a účasťou. Kardinalita vyjadruje maximálny počet partnerských inštancií a účasť minimálny. **Kardinalita** sa zadáva hodnotami 1, n a **účasť** hodnotami 0, 1. Preto početnostná charakterizácia vzťahu pri ER modelovaní je *hrubá*. Pripomíname, že presnejšie vyjadrenie početnosti môžeme dosiahnuť napr. v menej rozšírenom Object Role Modeling pomocou frekvencie a v SQL pomocou CONSTRAINT.

Tri základné typy vzťahov sú založené na kardinalite/počte

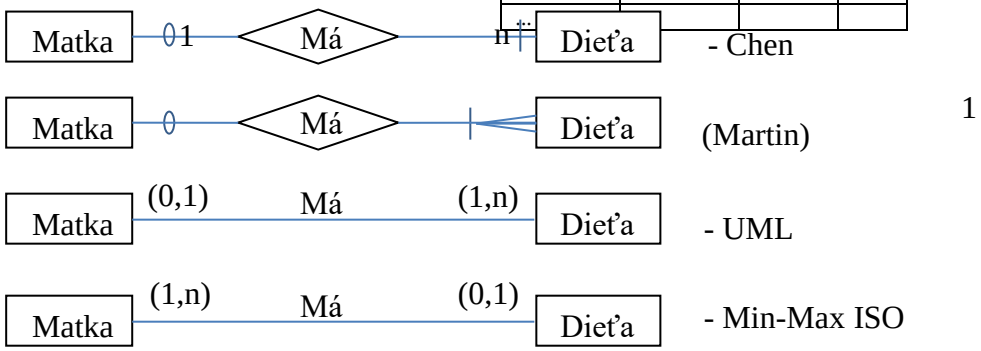
- **vzťah 1:1** – hovoríme, že medzi dvomi entitami je vzťah 1:1, ak inštancie jednej entity (jednému riadku tabuľky) zodpovedá jedna partnerská inštancia druhej entity (jeden riadok druhej tabuľky) v oboch smeroch. Príkladom je vzťah Osoba – Občiansky preukaz: jedna osoba má (maximálne) jeden OP a jeden OP patrí (práve) jednej osobe. Ďalší príklad je napr. Letenka – Sedadlo.
- **vzťah 1:n** - hovoríme, že medzi dvomi entitami je vzťah 1:n, ak inštancie jednej entity zodpovedá viac inštancií druhej entity, ale v druhom smere inštancie druhej entity zodpovedá jedna inštancia prvej entity. Príklad Matka – Dieťa pozri nižšie.
- **vzťah m:n** – v oboch smeroch ide o vzťah 1:n. Príkladom je Učiteľ – Predmet: jeden učiteľ môže učiť viac predmetov, daný predmet môže byť zabezpečený viacerými učiteľmi.

Daný *kardinálny typ* môžeme upresniť **účasťou**, ktorá má hodnotu **1** ak účasť je **nutná**. Žiaľ grafické diagramy kardinalitu a účasť neoznačujú jednotne. Vzťah Matka – Dieťa typu 1:n

ilustrujeme dvomi tabuľkami a štyrmi rôznymi ale ekvivalentnými diagramami, ktoré kardinalitu upresňujú pomocou účasti. Všimnime si, že prvá matka má dve deti a matka tretieho dieťaťa v evidencii je neznáma.

Matka		
idM	Meno	...
1		
2		
...		

Dieťa			
idD	Meno	idM	...
1		1	
2		2	
3		null	
4		1	
...			

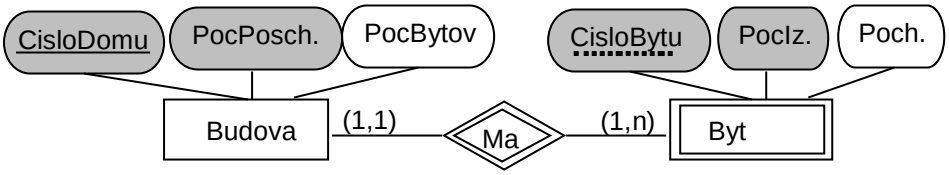


Rôzne označenia kardinality a účasti

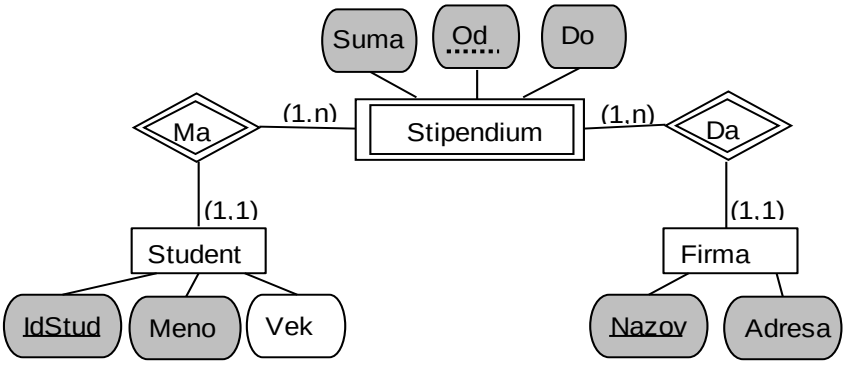
3) Slabé entity

Slabá entita nemôže byť jednoznačne identifikovaná vlastnými atribútmi – potrebuje k tomu aj primárny kľúč *rodiča*, *základnej entity*.

Budovy môžu mať byty s rovnakými atribútmi a s rovnakými číslami bytu.
Preto jednoznačná identifikácia bytu nie je možné bez toho, aby sme ju nespojili s budovou.
Budova je základná entita s jedinečným identifikátorom CisloDomu.
Byt je slabá entita, nemá vlastný jedinečný identifikátor.



Byty môžu mať rovnaké čísla v rôznych budovách - teda CisloBytu neidentifikuje Byt jednoznačne.

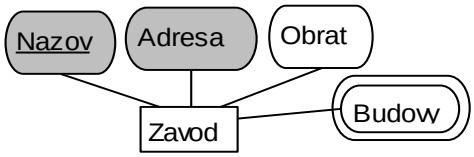


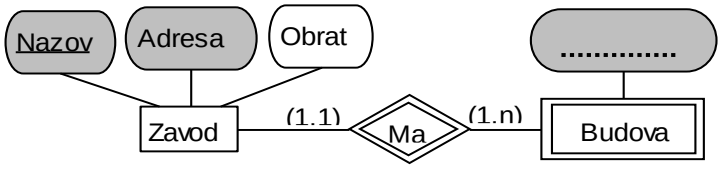
Štipendium má dvoch identifikujúcich *rodičov*: *Študent* a *Firma*.

Poznámka:
V databáze Poliklinika tabuľka NAVSTEVA by mohla byť definovaná ako slabá entita s primárnym kľúčom trojice (minuta, idL, idP).

4) Rozhodovania pri modelovaní

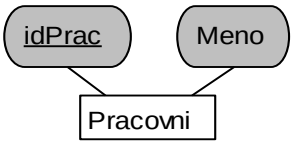
1) Náhrada (resolution) viachodnotových atribútov

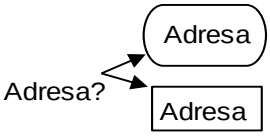




2) Náhrada vzťahu typu m:n, Pracovník - Projekt

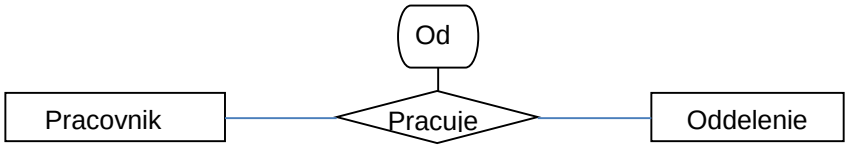
3) Môže sa stať, že sa má rozhodnúť, ako treba vlastnosť modelovať: ako atribút alebo ako entita



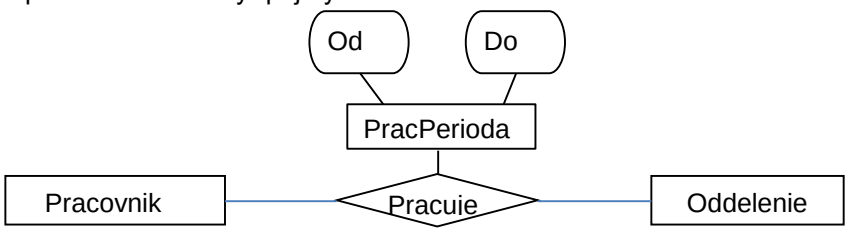


- viac adries pre pracovníka
- rozbyt adresu na atribúty kvôli dopytovaniu ako: "Nájdi pracovníkov z ulice Rožňavská"

4) Atribút vzťahu – v ER je to dovolené



Ak pracovník môže byť prijatý viackrát:



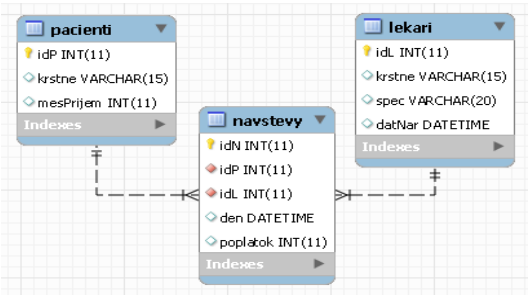
a) Reverse engineering v MySQL

Model/ERD z DB

Generovanie ERD:

Database ⇒ Reverse Engineer ⇒ Next ⇒ Psw ⇒ Next ⇒ Poliklinika

⇒ Next ⇒ Next ⇒ Execute ⇒ Next ⇒ Finish



ER Diagram Polikliniky

Voľba označenia vzťahu:

Model ⇒ Relationship Notation ⇒ Crow's Foot / UML

b) VZŤAH 1:1

Tabuľky Osoba a ObcianskyPreukaz
Vytvorte vzťah 1-1 pre Osoba a ObcianskyPreukaz

```
DROP Database IF EXISTS Osoba_OP;
CREATE Database IF NOT EXISTS Osoba_OP;
USE Osoba_OP;
#drop table op;
#drop table osoba;

create table osoba(
        idOs int not null primary key,
        meno varchar(20),
        rokNarodenia int
);
```

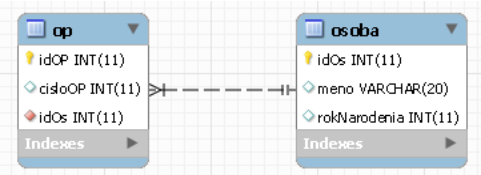
```
create table OP(
        idOP int not null primary key,
        cisloOP int,
        idOs int not null, --
        CONSTRAINT c1 FOREIGN KEY (idOs) REFERENCES Osoba (idOs)
);
insert osoba values(1,'F', 1993), (2,'P',1987), (3,'K',1976);
insert OP values(1,45,1), (2,46,3), (3,47,2);
```

Tabuľka OP nemôže obsahovať dva riadky s rovnakou hodnotou pre idOS:

```
-- insert OP values(4,48,2);  -- Chyba sa nehlasi - Osoba 2 nemoze mat 2 OP

select * from OP;

SHOW CREATE TABLE OP;
-- =>
#CREATE TABLE `op` (
#  `idOP` int(11) NOT NULL,
#  `cisloOP` int(11) DEFAULT NULL,
#  `idOs` int(11) NOT NULL,
#  PRIMARY KEY (`idOP`),
#  UNIQUE KEY `idOs` (`idOs`),
#  CONSTRAINT `c1` FOREIGN KEY (`idOs`) REFERENCES `osoba` (`idOs`)
#) ENGINE=InnoDB DEFAULT CHARSET=utf8 ...
```



Žiaľ, MySQL zatiaľ nevykresľuje správne ERD pre vzťah 1:1.

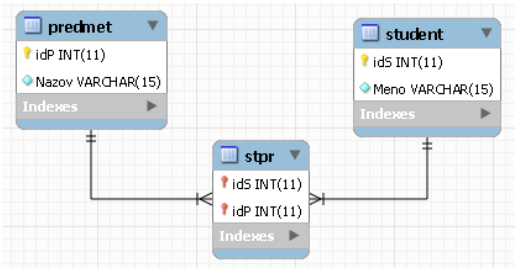
c) VZŤAH m:n

Tabuľky Student a Predmet a ich prepojovacia tabuľka

Vytvorte vzťah m:n pre Student a Predmet

```
DROP Database IF EXISTS Student_Predmet;
CREATE Database IF NOT EXISTS Student_Predmet;
USE Student_Predmet;

#drop table StudPred;
#drop table Predmet;
#drop table Student;
CREATE TABLE Student (
  idS int NOT NULL,
  Meno varchar(15) NOT NULL,
  PRIMARY KEY (idS)
);
CREATE TABLE Predmet (
  idP int NOT NULL,
  Nazov varchar(15) NOT NULL,
  # idS int NOT NULL,
  PRIMARY KEY (idP)
  # CONSTRAINT cc FOREIGN KEY (idS) REFERENCES Student (idS)
);
CREATE TABLE StudPred (
  idS int NOT NULL,
  idP int NOT NULL,
  CONSTRAINT c1 FOREIGN KEY (idS) REFERENCES Student (idS),
  CONSTRAINT c2 FOREIGN KEY (idP) REFERENCES Predmet (idP),
  PRIMARY KEY (idS, idP)
);
insert Student Values(1,'Fero'), (2,'Jano'), (3,'Stevo');
insert Predmet Values(11,'DBS'), (12,'C#'), (13,'PAZ');
-- insert StudPred Values(1,11), (1,12), (2,11), (2,11); -- ERROR vdaka PK (idS, idP)
insert StudPred Values(1,11), (1,12), (2,11);
select * from StudPred;
```



15 Best Database Design Tools
https://www.guru99.com/free-database-diagram-design-tools.html
https://vertabelo.com/

Barry Williams, Data Modeling by Example
https://modelarchive.databases.biz/downloads/Data_Modeling_by_Example_a_Tutorial.pdf
https://datamodels.databases.biz/
=>

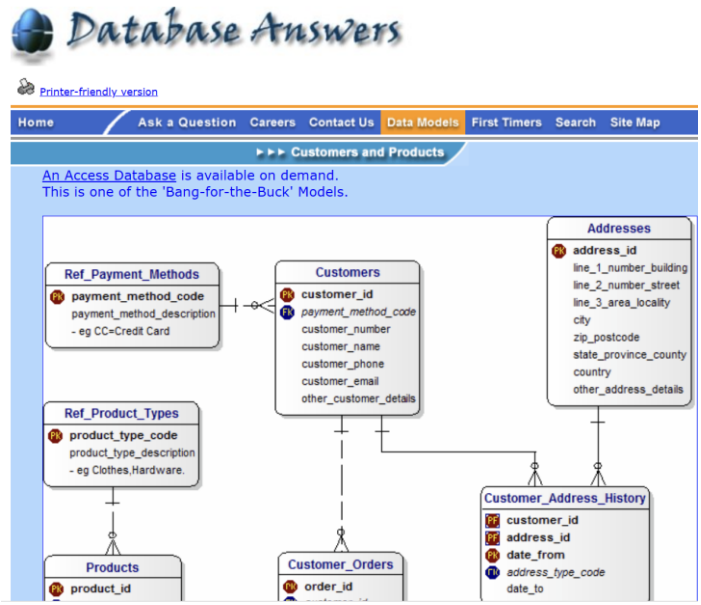
party)
6. Elephant Sanctuaries
7. Fiji Animals
8. Hedgehogs
9. Pitbulls and Parolees Platform
10. Turtle Conservancy
11. Zoo Feeding Time (5 minutes Tutorial)

8. **Assets**
1. Asset Management

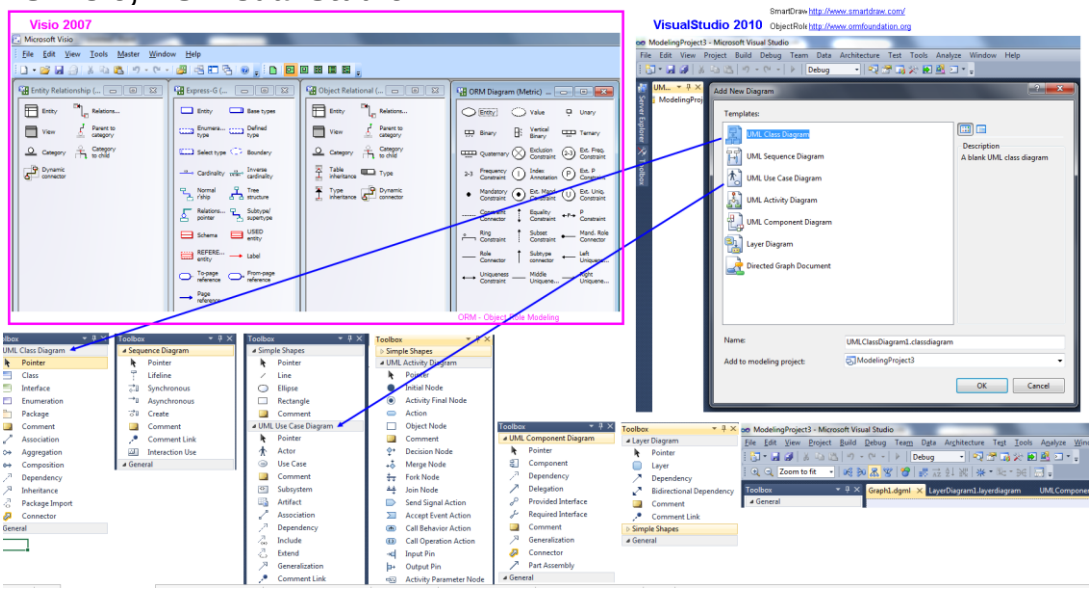
56. Customers and Price Comparison Web Sites
57. Customers and Products
58. Customers and Products (Canonical)
59. Customers and Products (Generic)
60. Customers and Products and Promotions

20. Student Self-Service Registration POC
21. Telecomms
22. UN Global Compact Platform
27. Product Regulatory Compliance Platform
28. Real Estate
29. Retail Data Platform
30. Retail Sales (POC)

=>



MS Visio, MS Visual Studio



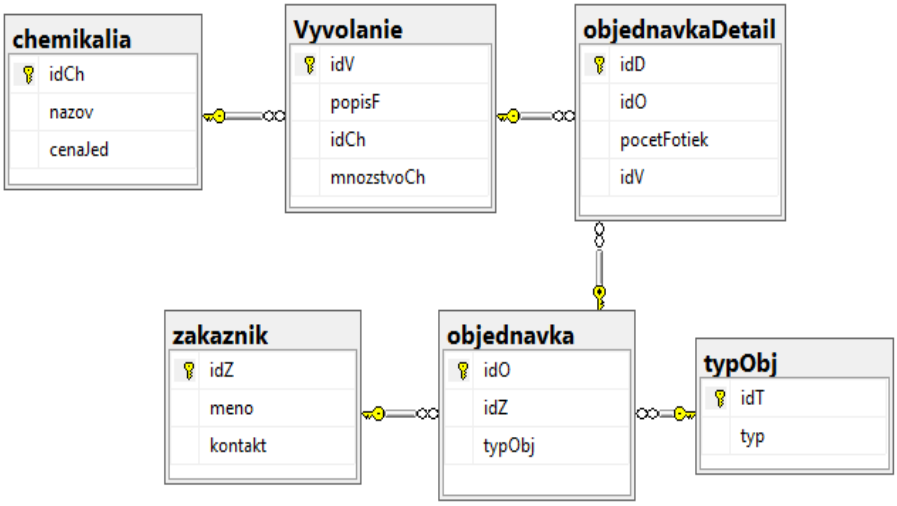
Niektoré systémy na kreslenie diagramov

Príklad. Fotolab

- Zákazník môže posielať alebo priniesť do FotoLabu filmy (**typ objednávky**).
- Zamestnanec prevezme ich - na objednávke určí (na základe *skutočného typu* filmu) aj **typ vyvolania**, čo znamená použitie konkrétnej chemikálie s konkrétnym množstvom.

a) Najprv predpokladáme, že zákazník naraz môže objednať vyvolanie viac filmov, ktoré sú ale *jedného typu*.

1 Zakaznik 2 TypObjednavky 3 Chemikalia
4 Objednavka 5 Vyvolanie
6 ObjednavkaDetail
(TypFilmu, TypVyvolania, ...)




```
drop database if exists Fotolab;
create database Fotolab;
use Fotolab;
create table zakaznik (
      idZ         int primary key,
      meno        varchar(20),
      kontakt     varchar(20)
);
create table typObj(
      idT         int primary key,
      typ         varchar(20)
);
create table chemikalia (
      idCh        int primary key,
      nazov       varchar(20),
      cenaJed     real
);
create table Vyvolanie (
      idV         int primary key,
      popisF      varchar(20),
      idCh        int,
```

```
     foreign key (idCh) references chemikalia(idCh),
     -- CONSTRAINT c1 FOREIGN KEY (idCh) REFERENCES
     chemikalia(idCh),
     mnozstvoCh int
);
create table objednavka (
     idO        int primary key,
     idZ        int,
     foreign key (idZ) references zakaznik(idZ),
     typObj     int,
     foreign key (typObj) references typObj(idT)
);
create table objednavkaDetail (
     idD        int primary key,
     idO        int,
     foreign key (idO) references objednavka(idO),
     pocetFotiek int,
     idV        int,
     foreign key (idV) references Vyvolanie(idV)
);
```

DÚ.

b) Predpokladajme, že filmy nemusia byť jedného typu.

Riešenie?

c) Vložte šesť objednávok do databázy. Kvôli cudzím kľúčom napĺňajte tabuľky v poradí, ako sú deklarované.

C) Vytvorenie databáz, tabuliek a integrita dát - 2

1) Integrita dát - 2
- **a) Narušenie a zabezpečenie integrity dát**
- **b) Obmedzenia**

2) Primárny a sekundárny kľúč - PK a FK, UNIQUE
- **Vytvorenie a odstránenie PK a FK, NOT NULL**
- **a) v rámci definície tabuľky**
- **b) pomocou ALTER TABLE**
  - **- bez CONSTRAINT => generovane meno**
  - **- pomocou CONSTRAINT s menom**
- **c) UNIQUE**
- **d) Kompozitný kľúč**
- **e) DEFAULT hodnota**

3) CHECK constraint, triggery
4) Spätná kontrola pri **ALTER TABLE**
5) … ADD / DROP COLUMN
6) Zistenie obmedzení programovo
7) ENUM obmedzenie
8) UPDATE a DELETE
9) CREATE TABLE ... SELECT / LIKE
10) Indexy

1) Integrita dát - 2
a) Narušenie a zabezpečenie integrity dát

Integrita (celistvosť, úplnosť a neporušenosť) dát je platnosť (validity) dát. Integrita dát sa zabezpečuje pomocou **obmedzení**. Modelovanie DB je v podstate modelovanie obmedzení na integritu, veď pravidlá reálneho sveta a princípov relačnej teórie transformujeme na DB model ako obmedzenia na integritu dát.

Príčiny porušenia integrity dát: - ľudská nepozornosť pri zadávaní údajov - chyba počas prenosu dát cez sieť - softvérové chyby - hardvérová porucha - infekcia vírusmi - prírodná pohroma.	Najdôležitejšie **kroky prevencie** straty dát: - systematické zálohovanie DB - bezpečnostné opatrenia na všetkých úrovniach - vhodné rozhranie s reštrikciami pre zadávanie dát - integrita dát.

Okrem troch základných typov

- **integrita entít** (jedinečnosť) => **entita** - relačná schéma; predmet/objekt/riadok

Entita, ako abstrakcia modelovanej časti sveta pri navrhovaní databáz, je vec, inštancia ktorej je schopná nezávislej existencie a ktorá môže byť jednoznačne identifikovaná. Konkrétna tabuľka je inštancie entity (relačnej schémy). Pripomíname, že namiesto entity a jej inštancie sa požívajú aj definície entitný typ a entita.
- **doménová integrita** (typ)
- **referenčná integrita** (cudzí kľúč)

je možné zaviesť aj iné **typy** integrity dát, ako:

- **používateľom definovaná integrita** http://msdn.microsoft.com/en-us/library/aa933058.aspx

b) Integrita dát a obmedzenia

Integrita entít definuje jedinečné riadky v rámci danej tabuľky **pomocou**
- id stĺpca, obmedzení `UNIQUE` alebo `PRIMARY KEY` a `auto_increment`

Doménová integrita znamená zabezpečenie platných hodnôt v danom stĺpci **pomocou**
- dátových typov a
- rozsahu hodnôt s použitím
    - definícií `NOT NULL` a `DEFAULT`
    - `CHECK` a `FOREIGN KEY` obmedzení

Referenčná integrita
- znamená referenciu iba na existujúce hodnoty a kaskádovitú zmenu (konzistentnosť) v referenciách v prípade zmeny hodnoty kľúča
- zabezpečuje prepojenosť tabuliek (ich vzťah) **pomocou**
    - `FOREIGN` a `PRIMARY` kľúčov
    - `FOREIGN` a `UNIQUE` kľúčov

Obmedzenia (Constraints)

	Int.entít	Doménova int.	Ref.int.
PRIMARY KEY	✓		✓
FOREIGN KEY		✓	✓
UNIQUE	✓		✓
CHECK		✓	
DEFAULT		✓	
NOT NULL		✓	

2) Primárny a sekundárny kľúč - PK a FK, UNIQUE

Primárny kľúč
- zabezpečuje jedinečnosť riadkov
- každá tabuľka iba 1 PK
- PK sa skladá z jednej alebo viac stĺpcov

Cudzí kľúč
- každá tabuľka môže mať viac FK
- FK sa skladá z jednej alebo viac stĺpcov

Poznámky:
- každá tabuľka by mala mať jeden/alebo viac kľúčov
- kľúč nie je lokalitou dát/riadkov (LetSem: B-strom = level nodes ∪ leaf nodes;
  node (uzol): page/stránka 8 kb;
  leaf nodes: data pages => data rows;
  level nodes: index pages => index rows)
- namiesto "nasledujúceho" a "predchádzajúceho riadku" by sme mali používať "ďalšieho". Treba rozlíšiť *fyzikálne* a *logické* (miesto).
- nie jednotlivé riadky, ale celé množiny riadkov vkladajme do, mažme z a zmeňme (update) v tabuľky/e.

Vytvorenie a odstránenie PK a FK, NOT NULL
a) V rámci definície tabuľky
b) pomocou ALTER TABLE
- bez CONSTRAINT => generované meno
- pomocou CONSTRAINT s menom
c) UNIQUE
d) Kompozitný kľúč
e) DEFAULT hodnote

a-b) PK v rámci definície tabuľky alebo v ALTER TABLE

Síce sa dá vytvoriť meno pre `CONSTRAINT`, ale to pri hlásení chyby s PK sa nezobrazí. Pri PK `not null` nie je nutné.

```
use dbmaz;
drop table if exists T;
CREATE TABLE T(id int, x int);
#CREATE TABLE T(id int not null Primary key, x int);
#CREATE TABLE T(id int Primary key, x int);
#CREATE TABLE T(id int, x int, CONSTRAINT PKid Primary key (id));
ALTER TABLE T ADD CONSTRAINT PKid Primary key (id);
#ALTER TABLE T ADD Primary key (id);
insert T values(0,10),(1,11); -- OK, no error pri PK
#insert T values(1,12);       -- zapricinuje error pri PK
select * from T;
```

PK kľúč nemusí mať typ `int`:

```
Drop table if exists T;
CREATE TABLE T(idT CHAR(15) NOT NULL PRIMARY KEY);
INSERT T VALUES('Abara');
```

AUTO_INCREMENT `id`
V `INSERT` netreba uviesť `id` ani jeho hodnotu:

```
Drop table if exists T;
CREATE TABLE T(id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
               y int);
INSERT T(y) VALUES(12); # 1 12
```

a-b) FK v rámci definície tabuľky alebo v ALTER TABLE

Bez `CONSTRAINT` sa automaticky vygeneruje meno pre FK. V prípade chyby s cudzím kľúčom sa v chybovom hlásení meno `CONSTRAINT` sa objaví – na rozdiel od PK.

```
USE DBmaz;
Drop table if exists U;
Drop table if exists T;
CREATE TABLE T(idT int NOT NULL PRIMARY KEY, x INTEGER);
INSERT T VALUES(1,11),(2,12);

CREATE TABLE U(idU int PRIMARY KEY, ckT int, a CHAR(5)
     #, FOREIGN KEY (ckT) REFERENCES T (idT)
     #, CONSTRAINT FKckT FOREIGN KEY (ckT) REFERENCES T (idT)
);
#ALTER TABLE U ADD FOREIGN KEY (ckT) REFERENCES T (idT);
ALTER TABLE U ADD CONSTRAINT FKckT FOREIGN KEY (ckT) REFERENCES T
(idT);
INSERT U VALUES(11,1,'a'),(12,1,'a');
INSERT U VALUES(13,3,'c');   # error
```

Ako odstrániť FK a CONSTRAINT?

```
drop database if exists DBmaz;  create database DBmaz;  use DBmaz;
Drop table if exists U;  Drop table if exists T;
CREATE TABLE T( idT CHAR(15) NOT NULL PRIMARY KEY );
CREATE TABLE U( ckT CHAR(15)) ENGINE = INNODB;
ALTER TABLE U ADD CONSTRAINT k1 FOREIGN KEY(ckT) REFERENCES T (idT);
SHOW CREATE TABLE U;
-- ALTER TABLE U DROP CONSTRAINT k1;
ALTER TABLE U DROP FOREIGN KEY k1;
SHOW CREATE TABLE U;
```

c) UNIQUE - jedinečnosť

- zabezpečuje jedinečnosť hodnôt v *nekľúčovom* stĺpci
- na rozdiel od PK v tabuľke môže byť viac UNIQUE obmedzení
- `unique` dovolí viac (MS SQL Server iba jednu) *null* hodnôt
- cudzí kľúč sa môže odvolávať na UNIQUE stĺpec

```
DROP TABLE IF EXISTS H;
CREATE TABLE H (id INT PRIMARY KEY, x INT UNIQUE);
INSERT H VALUES(1,8);
-- INSERT H VALUES(2,8);     -- NO duplicate
-- INSERT H VALUES(NULL,8); -- NO cannot be null
INSERT H VALUES(2,NULL);
INSERT H VALUES(3,NULL);
SELECT * FROM H limit 3;
```

id	x
2	NULL
3	NULL
1	8
NULL	NULL

Čo sa stane, ak sa cudzí kľúč odvoláva na nie PRIMARY KEY, nie UNIQUE stĺpec?

d) Kompozitný kľúč

Vytvorenie kompozitného kľúča z viacerých atribútov – zo stĺpcov `i` a `j`

```
DROP TABLE IF EXISTS t6, t5,t4,t3;
CREATE TABLE T3(i INT NOT NULL PRIMARY KEY, j INT);
CREATE TABLE T4(i INT, j INT, k INT, PRIMARY KEY(i,j));
CREATE TABLE T5(i INT NOT NULL, j INT NOT NULL, k INT, PRIMARY
KEY(i,j));
CREATE TABLE T6(i INT NOT NULL, j INT NOT NULL, k INT);
ALTER TABLE T6 ADD CONSTRAINT pk_T2 PRIMARY KEY(i,j);
```

Je obmedzenie NOT NULL nutné?

e) DEFAULT

```
    x VARCHAR(25) DEFAULT 'Polozka xyz';
```

Ak sa v definícii stĺpca neuvádza `NOT NULL`, potom ide o `DEFAULT NULL`.

```
DROP TABLE IF EXISTS TT;
CREATE TABLE TT(x VARCHAR(11) DEFAULT 'Polozka xyz', y INT NOT NULL,
z INT DEFAULT NULL);
-- <=>
#CREATE TABLE TT(x VARCHAR(10) DEFAULT 'Polozka xyz', y INT NOT
NULL, z INT);
INSERT TT(y) VALUES(3);
SELECT * FROM TT;  # Polozka xyz, 3, NULL
```

3) CHECK constraint, triggery a ENUM

http://dev.mysql.com/doc/refman/5.1/en/create-table.html

CHECK obmedzenie slúži na zabezpečenie **doménovej** integrity. CHECK môžeme použiť buď v CREATE TABLE alebo ALTER TABLE a buď s CONSTRAINT alebo bez.

Pravidlá a výhody CHECK obmedzení:
- základné obmedzenia sú na jednom mieste
- tabuľky a stĺpce môžu obsahovať viac CHECK obmedzení
- ich nie je možné pomocou dopytu alebo aplikačného programu (okrem ALTER) prepísať
- optimalizačné programy pre dopyt, INSERT, UPDATE, DELETE analyzujú obsah CHECK obmedzení
- CHECK obmedzenie je možné pomenovať
- názov pomenovaného obmedzenia sa objaví v chybových hláseniach

```
use dbmaz;
-- a) P key
drop table if exists jaj;
#CREATE TABLE jaj(id int NOT NULL, x int);
#CREATE TABLE jaj(id int NOT NULL, CHECK (id>10), x int);
#CREATE TABLE jaj(id int NOT NULL, x int, CHECK (id>10));
CREATE TABLE jaj(id int NOT NULL, x int, CONSTRAINT viacAko10 CHECK
(id>10));
#⇔
#ALTER TABLE jaj ADD CONSTRAINT viacAko10 CHECK (id>10);
#ALTER TABLE jaj ADD CHECK (id>10);
insert jaj values(11,1); -- OK
insert jaj values( 0,1); -- hlasi sa chyba kvoli CHECK
select * from jaj;
```

CHECK nemôže spájať dva stĺpce
Napr.

```
DROP TABLE IF EXISTS T;
CREATE TABLE T (x int NOT NULL PRIMARY KEY,
                y int CONSTRAINT CK_2b CHECK (x<y));
```

Okrem CHECK môžeme použiť aj **triggery**.(na TRIGGER iba dve ukážky)

Pr.1 Hodnota y má byť menšia alebo rovná ako 100:

```
use dbmaz;
Drop table if exists T;
CREATE TABLE T(
  idT CHAR(15) NOT NULL PRIMARY KEY,
  x INTEGER,
  y INTEGER NOT NULL CHECK(y<=100)
);
INSERT T VALUES('Kosice',NULL, 10);
#INSERT T VALUES('Poprad', 3, 101); # error
SELECT * FROM T;
```

Pr.2 Pre y väčšia hodnoty ako 100 sa redukujú na 100:

```
DROP TRIGGER IF EXISTS trig_T;
DELIMITER $$
CREATE TRIGGER trig_T BEFORE INSERT ON T
FOR EACH ROW
BEGIN
  IF NEW.y>100 THEN SET NEW.y=100;
  END IF;
END
$$ DELIMITER ;

INSERT T VALUES ('Moson', 4, 161); #100
SELECT * FROM T;
```

idT	x	y
Kosice	NULL	10
Moson	4	100
NULL	NULL	NULL

Pr.3 Súhra medzi dvomi stĺpcami x, y (pokračovanie T2)

```
DROP TRIGGER IF EXISTS trig_T2;
DELIMITER $$
CREATE TRIGGER trig_T2 BEFORE INSERT ON T
FOR EACH ROW
BEGIN
  IF NEW.x > NEW.y THEN SET NEW.y=0;
  END IF;
END
$$ DELIMITER ;

INSERT T VALUES ('Roznava', 5, 1); # => 0
SELECT * FROM T;
```

idT	x	y
Kosice	NULL	10
Moson	4	100
Roznava	5	0
NULL	NULL	NULL

4) Spätná kontrola pri ALTER TABLE

Spätná kontrola pri **ALTER** table sa vykoná nielen pre PK, ale aj pre CHECK obmedzenie. Pozor, aj bez odstránení konfliktových hodnôt sa ide ďalej.

a) PK

```
use dbmaz;
drop table if exists jaj;
CREATE TABLE jaj(id int not null, x int);
insert jaj values(0,1);
insert jaj values(0,2);
select * from jaj;
```

Hlási sa chyba kvôli spätnej kontrole `PRIMARY`:

```
ALTER TABLE jaj ADD CONSTRAINT pkid Primary key (id);
```

b) CHECK

Aj tu pri `ALTER` sa hlási chyba pri spätnej kontrole `CHECK`:

```
-- Pokracovanie:
ALTER TABLE jaj ADD CONSTRAINT viacAko10 CHECK (id>10);
```

5) … ADD / DROP COLUMN

```
ALTER TABLE jaj ADD COLUMN xxx int;
select * from jaj;
ALTER TABLE jaj DROP COLUMN xxx;
select * from jaj;
```

6) Zistenie obmedzení programovo

Uvažujme kód:

```
USE DBmaz;
Drop table if exists U;
Drop table if exists T;
CREATE TABLE T (idT CHAR(15), x INTEGER);
#ALTER TABLE T ADD CONSTRAINT PKidT Primary key (idT);
ALTER TABLE T ADD Primary key (idT);
ALTER TABLE T ADD CONSTRAINT CKx CHECK (idT<x);
INSERT T VALUES('18',19); -- OK
#INSERT T VALUES('19',19); -- error

CREATE TABLE U( idU int, ckT CHAR(15) not null, y int) engine
innodb;
ALTER TABLE U ADD CONSTRAINT PKidU Primary key (idU);
ALTER TABLE U ADD CONSTRAINT FKckT FOREIGN KEY (ckT) REFERENCES T
(idT);
ALTER TABLE U ADD CONSTRAINT CKy CHECK (idU<y);
INSERT U VALUES(1,18,3);
#INSERT U VALUES(2,17,3); -- error
SELECT * FROM T;
SELECT * FROM U;
```

Na zistenie **mena** `CONSTRAINTu` (pozri `CKx`) sa dá použiť buď príkaz

```
SHOW CREATE TABLE T; -- =>Open value in viewer myš - pravé tlačidlo
##DESCRIBE T;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Table	Create Table
T	CREATE TABLE `t` (`idT` char(15) NOT NULL, `x` int DEFAULT NULL, PRIMARY KEY (`idT`), CONSTRAINT `CKx` CHECK ((`idT` < `x`))) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

alebo tabuľku KEY_COLUMN_USAGE z databázy pohľadov INFORMATION_SCHEMA
http://dev.mysql.com/doc/refman/5.6/en/information-schema.html

```
select TABLE_NAME,COLUMN_NAME, CONSTRAINT_NAME,
         REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME
      from INFORMATION_SCHEMA.KEY_COLUMN_USAGE
      where TABLE_SCHEMA = "dbmaz" and TABLE_NAME = "U"
            and referenced_column_name is not NULL;
# pozri FKckT
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TABLE_NAME	COLUMN_NAME	CONSTRAINT_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
u	ckT	FKckT	t	idT

```
ALTER TABLE U DROP FOREIGN KEY FKckT;
```

Pozri nižšie DESCRIBE Tab pri indeksoch.

7) ENUM obmedzenie

```
DROP TABLE IF EXISTS Vykon;
CREATE TABLE Vykon (
    vykon int,
    size ENUM('Pon', 'Uto', 'Str', 'Stv', 'Pia')
);
INSERT Vykon VALUES(1,'Pon'), (5,'Str');
SELECT * from Vykon;
```

vykon	size
1	Pon
5	Str

8) CREATE TABLE ... SELECT / LIKE

Kým

```
CREATE TABLE Vyk2(...) SELECT * FROM Vykon;
```

vkladá celý obsah tabuľky Vykon do práve vytvorenej prázdnej tabuľky Vyk2 pomocou ... , príkaz

```
CREATE TABLE Vyk3 LIKE Vykon;
```

vytvorí iba kópiu Vyk3 schémy tabuľky Vykon bez vkladania riadkov.

Pokračovanie kódu po ENUM obmedzení tabuľky Vykon.

```
DROP TABLE IF EXISTS Vyk2;
CREATE TABLE Vyk2  (
     vykon int,
    size VARCHAR(3)) SELECT * FROM Vykon;
INSERT Vyk2 VALUES(7,'Str');
SELECT * from Vyk2;
```

vykon	size
1	Pon
5	Str
7	Str

```
DROP TABLE IF EXISTS Vyk3;
CREATE TABLE Vyk3 LIKE Vykon;
SHOW CREATE TABLE Vyk3;
```

Table	Create Table
Vyk3	CREATE TABLE `vyk3` (`vykon` int(11) DEFAULT NULL, `size` enum('Pon','Uto','Str','Stv','Pia') DEFAULT NULL) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

```
INSERT Vyk3 VALUES(7,'Str');
SELECT * from Vyk3;
```

vykon	size
7	Str

9) UPDATE a DELETE

a) Edit -- Preferences -- SQL Editor -- dole odfajknúť Safe Updates
b) Query -- Reconnect to Server -- Exit => Prihlásiť sa

alebo nastaviť `SQL_SAFE_UPDATES` na 1 alebo 0 (1⇔ON - bezpečné).

Pokračovanie.

```
SHOW VARIABLES LIKE 'SQL_SAFE_UPDATES';
```

Variable_name	Value
sql_safe_updates	ON

```
SET SQL_SAFE_UPDATES = 0; # 1 => nasl.prik. error
UPDATE Vyk2 SET vykon = 22 WHERE size='Pon';
#UPDATE Vyk2 SET vykon =  1 WHERE size='Pon';
SELECT * from Vyk2;
```

vykon	size
22	Pon
5	Str
7	Str

```
DELETE FROM Vyk2 WHERE size='Pon';
SELECT * from Vyk2;
```

vykon	size
5	Str
7	Str

```
UPDATE Vyk2 SET vykon = 1;  -- Bez WHERE nebezpecne!?
DELETE FROM Vyk2;           -- Bez WHERE nebezpecne!?
```

Obnovenie stavu pred poslednými dvomi nebezpečnými kódmi:

```
SET SQL_SAFE_UPDATES = 0;
DROP TABLE IF EXISTS Vyk2;
CREATE TABLE Vyk2  (
     vykon int,
    size VARCHAR(3)) SELECT * FROM Vykon;
INSERT Vyk2 VALUES(7,'Str');
DELETE FROM Vyk2 WHERE size='Pon';
SELECT * from Vyk2;

SET SQL_SAFE_UPDATES = 1;
UPDATE Vyk2 SET vykon = 1; -- error OK veď nebezpečné
```

10) Indexy (iba ukážky)

Indexy zrýchlia vyhľadanie, lokalizáciu údajov. **Podrobnejšie v LS**.

```
# drop database if exists DBmaz;
# create database DBmaz;
use DBmaz;
DROP TABLE IF EXISTS T;
DROP TABLE IF EXISTS U;
CREATE TABLE T( idT CHAR(15) NOT NULL PRIMARY KEY );
CREATE TABLE U( idU int NOT NULL PRIMARY KEY,
                ckT CHAR(15), FOREIGN KEY (ckT) REFERENCES T (idT));
```

Tu iba získame informácie:

```
DESCRIBE U;
```

Field	Type	Null	Key	Default	Extra
idU	int(11)	NO	PRI	NULL	
ckT	char(15)	YES	MUL	NULL	

```
SHOW INDEX FROM U;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment
u	0	PRIMARY	1	idU	A	0	NULL	NULL		BTREE	
u	1	ckT	1	ckT	A	0	NULL	NULL	YES	BTREE	

```
EXPLAIN SELECT * FROM U;
EXPLAIN SELECT * FROM U WHERE U.idU > 5;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	U	NULL	index	NULL	ckT	61	NULL	1	100.00	Using index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	U	NULL	index	PRIMARY,ckT	ckT	61	NULL	1	100.00	Using where; Using index