# **MySQL**

Collation -> utf8mb4\_unicode\_ci

Import -> pridam databazu Export -> Custom -> save to output file -vytvorim databazovy subor

<https://db57.websupport.sk/> //stranka s realnym prihlasenim

user: Dani pass: Dani12

**SELECT** \* **FROM** directors //vsetko z tabulky directors

SELECT \* FROM movies **WHERE** director\_id = 1 //s podmienkou s id = 1

SELECT \* FROM movies **WHERE** director\_id = 2 **AND** year >= 2000 **AND** year <= 2010

//s viacerymi podmienkami

SELECT \* FROM movies WHERE director\_id = 2 AND year **BETWEEN** 2000 AND 2010

//viacero podmienok inym sposobom

SELECT \* FROM movies WHERE director\_id **IN** (2, 3) AND year BETWEEN 2000 AND 2010

//ak chcem vymenovat viacero ID

SELECT \* FROM movies WHERE director\_id IN (2, 3) AND year BETWEEN 2000 AND 2010

AND year **!=** 2004 //s vynimkou roku 2004

SELECT first\_name, last\_name, **YEAR**(birthdate), **MONTH**(birthdate), **DAY**(birthdate)

FROM directors //mozem konkretizovat, pokial mam v database datumy

SELECT first\_name, last\_name FROM directors WHERE **YEAR**(birthdate) BETWEEN 1950 AND 1960 //funkcia v podmienke

SELECT NOW() //zistim aktualny datum a cas

SELECT first\_name, last\_name, country, YEAR(**NOW**()) - **YEAR**(birthdate) **AS** Age

FROM directors //kolko ma rokov, dane policko nazvem ako Age

SELECT first\_name, last\_name, country, YEAR(NOW()) - YEAR(birthdate) AS age, **CONCAT**(first\_name, ' ', last\_name) AS name //pri vytahovani z databazy mozem spojit mena FROM directors ORDER BY age ASC //zoradene zostupne alebo vzostupne(DESC)

SELECT \* FROM movies **LIMIT** 5 **OFFSET** 10

//limit znamena pocet buniek a offset od ktorej bunky zaciname, pre toto je to id od 10 po 15

SELECT **MAX**(gross) FROM movies //maximalna hodnota z gross(MIN, AVG- priemer)

SELECT **FORMAT**(MAX(gross), 2) FROM movies //naformatovane na 2 desatinne miesta

SELECT CONCAT('$', FORMAT(MAX(gross), 0)) AS gross FROM movies //prilepim $

SELECT title, year, CONCAT('$', FORMAT(gross, 0)) as **total\_gross** FROM movies ORDER BY **gross** DESC //davat pozor podla coho zoradujem, total gross mam ako string uz

**INSERT INTO** directors (first\_name, last\_name, country, birthdate) **VALUES** ('Paul Thomas', 'Anderson', 'United States', '1970-07-26')

//vlozenie hodnoty do tabulky directors, v zatvorke su stlpce a hodnoty v rovnakom poradi

**INSERT INTO** directors **VALUES** (NULL, 'Ingmar', 'Bergman', 'Sweden', '1918-07-14')

//nemusim vymenovat stlpce, ale v tom pripade musim zadat vsetky stlpce, null je miesto ID,

ktore chcem aby sa predvyplnilo same

**INSERT INTO** directors **VALUES** (DEFAULT, 'Andrei', 'Tarkovsky', 'Russia', '1932-04-04')

//ak nefunguje NULL, mozem pouzit aj slovo DEFAULT

SELECT \* FROM directors WHERE country IS **NOT NULL**

//vyberie nenulove hodnoty z daneho stlpca(directors)

SELECT \* FROM directors WHERE country IS **NOT NULL** AND country **!= ''**

//ja viem nastavit, ci dane hodnoty v stlpci mozu byt null alebo nie,pokial nie, tak tam bude davat prazne policko, preto niekde moze byt null a niekde zas prazdne policko, tymto sposobom dam zobrazit policka, ktore su komplet vyplnene

SELECT \* FROM directors WHERE **NULLIF** (country, '') IS NULL

//NULLIF je funkcia ktora nastavi NULL, ak ma country hodnotu ‘’

INSERT INTO movies (director\_id, title, year, gross)

VALUES (9, 'The Wind Will Carry Us', 2000, 259510),

(9, 'Certified Copy', 2011, 1373975),

(9, 'Like Someone in Love', 2013, 239056) //vlozenie viacerych hodnot oddelit ciarkou

**UPDATE** directors **SET** country = 'Poland' WHERE id = 8

//update 8. Riadku

**UPDATE** directors **SET** country = 'United States' WHERE id **IN** (2, 3, 5)

//update viacerych riadkov

UPDATE directors SET country = 'Iran' WHERE id = (

SELECT director\_id FROM movies WHERE title = 'Dead Man’)

//upravi riadok s id, ktore sa rovna urcitej podmienke

UPDATE directors SET first\_name = 'Andrej', last\_name = 'Tarkovskij', country = 'Russia'

WHERE id = 7 //upravi viacero riadkov

INSERT INTO directors VALUES (4, 'Leos', 'Carax', 'France', '1960-11-22') **ON DUPLICATE KEY**

UPDATE country = 'France' //najprv vlozi, ak sa nepodari, tak spravi update

**DELETE** **FROM** movies WHERE id = 12 //mazanie

DELETE FROM movies ORDER BY id DESC LIMIT 1 //mazanie posledneho riadku  
SET SQL\_SAFE\_UPDATES = 0; //samotne delenie vypisovalo chybu, preto treba este toto   
 spravit

DELETE FROM movies **TRUNCATE** TABLE directors  
 //truncate je rychlejsi sposob mazania, pricom sa autoincrement nastavi od 1 a neda sa

niekedy mazat, ak dame podmienku nejaku, kde su poprepajane rozne tabulky

//delete from je pomalsi, s podmienkami nema problem, ale nenastavi autoincrement od 1

//obe mazu len obsah tabulky, nie samotnu tabulku

**DROP** TABLE IF EXISTS directors //vymazanie samotnej tabulky s nejakou podmienkou

**CREATE** TABLE `directors` (

`id` int(11) unsigned NOT NULL AUTO\_INCREMENT,

`director\_id` int(11) DEFAULT NULL,

`title` varchar(200) DEFAULT NULL,

`year` year(4) DEFAULT NULL,

`gross` decimal(15,2) DEFAULT NULL,

`genre` varchar(60) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `director\_id` (`director\_id`)

) //vytvorenie nejakej tabulky

SELECT \* FROM directors WHERE country **LIKE** 'United%'

//ak pridam LIKE, mam dostupne ‘%’, ktore sluzi na lubovolne doplnenie textu za United, cize

mi najde napr. United States alebo United Kingdom

SELECT \* FROM directors WHERE first\_name **LIKE** 'A%' OR first\_name **LIKE** 'J%'

//prve zaciatocne pismena na A alebo J

SELECT \* FROM movies WHERE title **LIKE** '%Avengers'

//vsetko co konci na Avengers

SELECT \* FROM movies WHERE title **LIKE** '%World%'

//niekde v texte je World, da sa pouzit aj pre DELETE aj UPDATE

SELECT **COUNT**(1) FROM movies //zrata mi vsetky riadky, je jedno co dam do zatvoriek

SELECT first\_name, last\_name, CONCAT('$', FORMAT(MAX(gross), 0)) AS max,

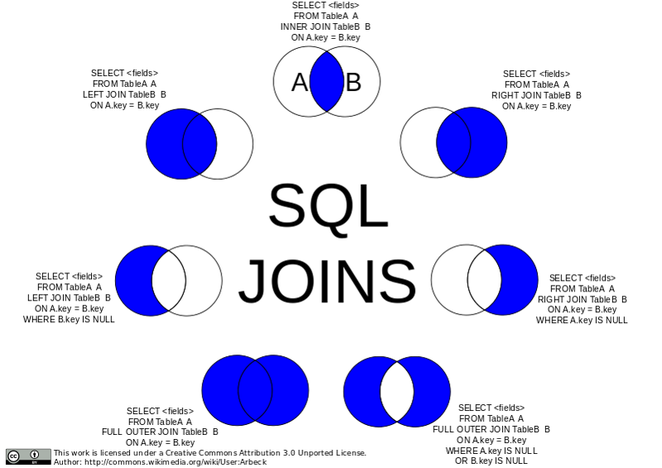
CONCAT('$', FORMAT(MIN(gross), 0)) AS min

FROM movies **JOIN** directors **ON** directors.id = director\_id WHERE director\_id = 4

//prepojenie dvoch tabuliek, za ON ide podmienka(aj viac), cez ktoru su tabulky prepojene, cez bodku pristupujem k jednotlivym stlpcom z tabulky, tento join vybere len tie zaznamy z tabulky, ktore su v oboch tabulkach(Prienik)

SELECT CONCAT(first\_name, ' ', last\_name) AS name, title, year FROM directors **LEFT JOIN** movies ON directors.id = movies.director\_id

//**LEFT JOIN** je rozdielny od JOIN tym, ze z lavej tabulky(za FROM) vyberie aj tie zaznamy, ktore sa nenachadzaju v pravej tabulke(za JOIN)(Zjednotenie), opacny sposob je pouzitie **RIGHT JOIN, OUTER JOIN** v MySQL nefunguje, ale v inej databaze vrati zjednotenie LEFT a RIGHT JOIN



SELECT CONCAT(first\_name, ' ', last\_name) AS name, title, year FROM directors LEFT JOIN movies ON directors.id = movies.director\_id

**UNION**

SELECT CONCAT(first\_name, ' ', last\_name) AS name, title, year FROM directors RIGHT JOIN movies ON directors.id = movies.director\_id

//nasimulovany OUTER JOIN(tento prikaz nefunguje v MySQL), cize RIGHT a LEFT JOIN dokopy pomocou **UNION**(spaja), ktory musi mat rovnaky pocet stlpcov na oboch stranach

SELECT d.\*, m.title, m.year FROM directors **AS d** RIGHT JOIN movies **AS m** ON d.id = m.director\_id //premenovane tabulky pomocou **aliasu** AS

SELECT d.\*, m.title, m.year FROM directors **d** RIGHT JOIN movies **m** ON d.id = m.director\_id //dokonca **alias** funguje aj bez AS, nemusim ho tam pisat

DELETE **d, m** FROM directors AS d RIGHT JOIN movies AS m ON d.id = m.director\_id WHERE d.id IS NULL //toto je **MULTI DELETE**, lebo mazem z viacerych tabuliek(mozem aj z jednej) naraz, pri nom musim za DELETE napisat, z ktorych tabuliek chcem mazat

SELECT genre, SUM(gross) FROM movies **GROUP BY** genre

//zgrupenie podla zanru, kazdy zaner bude vypisany len raz, ale pod nazvom sa moze nachadzat vacsi pocet filmov(zaznamov), takto mozem zistit zarobok kazdeho zanru

SELECT **DISTINCT** genre FROM movies //DISTINCT robi to iste co GROUP BY, ale ked sa zaner opakuje, tak dalsi ignoruje a negrupi ich, len vypise obsahujuce zanre po jednom

SELECT directors.first\_name, directors.last\_name, COUNT(1) as movies, CONCAT('$', FORMAT(MAX(movies.gross), 0)) AS max, CONCAT('$', FORMAT(MIN(movies.gross), 0)) AS min, CONCAT('$', FORMAT(AVG(movies.gross), 0)) AS avg

FROM movies JOIN directors ON directors.id = movies.director\_id

[ak by bola podmienka WHERE, nachadzala by sa tu]

**GROUP BY** director\_id **HAVING** movies=3 ORDER BY AVG(gross) DESC LIMIT 1

//ak chcem vybrat udaje pomocou podmienky WHERE a tieto udaje boli vytvorene groupenim, tak WHERE nebude mozne pouzit, treba pouzit HAVING a pisat ho az za GROUP BY, sucasne vidime aj **poradie prikazov** ak ich je viac

CREATE **TRIGGER** posts\_create BEFORE INSERT ON `posts`

FOR EACH ROW **SET NEW.created\_at = NOW(), NEW.updated\_at = NOW()**;

//trigger posts\_create, trigger vykona nejaku cinnost pri INSERTE napr. v nasom pripade , samotny kod triggeru je zvyrazneny

CREATE **TRIGGER** posts\_update BEFORE UPDATE ON `posts`

FOR EACH ROW **SET NEW.updated\_at = NOW(), NEW.created\_at = OLD.created\_at**;

//trigger posts\_update spravi trigger vzdy pri update, create necha na starej hodnote OLD

CREATE TABLE `**posts\_tags**` ( `**post\_id**` int(11) unsigned NOT NULL DEFAULT '0',

`**tag\_id**` int(11) unsigned NOT NULL DEFAULT '0', PRIMARY KEY (`tag\_id`,`post\_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

//tato tabulka sa nazyva **pivot**, pretoze prepaja dve tabulky s relaciou M:N, stlpce primarnych klucov su post\_id a tag\_id

SELECT t.tag FROM posts p JOIN posts\_tags pt ON (p.id = pt.post\_id) JOIN tags t ON (t.id = pt.tag\_id) WHERE p.id = 1 //vytiahnutie udaju pomocou pivota

SELECT p.\*, **GROUP\_CONCAT**(t.tag **SEPARATOR** '~||~') AS tags FROM posts p

JOIN posts\_tags pt ON (p.id = pt.post\_id) JOIN tags t ON (t.id = pt.tag\_id)

WHERE p.id = 1 **GROUP BY** p.id

//**GROUP\_CONCAT** vie zgrupit stringy a dat do jednej z jednotlivych tabuliek, takto mam pristup ku vsetkym tagom nejakeho komentu pomocou len jedneho query, separator je oddelovac jednotlivych stringov

SELECT m.id, title,CONCAT(first\_name, ' ', last\_name) AS name,

**GROUP\_CONCAT**(g.name) AS genre, year, gross

FROM movies m

JOIN directors d ON d.id= director\_id JOIN movies\_genres mg ON movie\_id = m.id

JOIN genres g ON g.id = genre\_id **GROUP BY** m.id

//este raz vysvetlenie GROUP BY a GROUP\_CONCAT:

//**GROUP BY** mi spoji hodnoty, v tomto pripade podla m.id, cize ak bude nejaky film s dvoma zanrami, bol by v tabulke dvakrat s id aj s title, len zaner by bol iny, po GROUP BY by sa tam nachadzal raz a aj ked tam bude len jeden zaner vypisany, v nom sa nachadza aj iny, len ho nevidime, pretoze je zgrupeny(ako keby tam bolo pole hodnot v jednom riadku)

//**GROUP\_CONCAT**(nastaveny na zanre g.name) sluzi zas na to, aby sme vedeli vypisat vsetky tieto zanre pre dany film, cize mi zobrazi skryte aj hodnoty vytvorene pomocou GROUP BY do jedneho riadku

**INDEXY**: urychluju hladanie z databazy, pri hladani nejdu riadok po riadku, ale pozru sa najprv na index, ktory bude odkazovat na urcity riadok v tabulke a ten rovno najde, treba si to predstavit ako knihu, ktora by na konci mala napisane slova s cislami stran, kde sa nachadza dane slovo. Index moze byt typu **HASH**, ktore funguje tak, ako som to popisal s knihou, do kluca ulozi dane slovo a ako hodnotu vrati pointer nan, da sa pouzit len na konkretne vyhladavanie, typu ci sa tam nachadza (ano/nie), nedalo by sa vykonavat matematicke operacie napr <, > atd podla nejakej hodnoty.

Typ indexu **BTREE**  sa da narozdiel od predosleho usporiadat, vazit, porovnavat, funkcia sa da prirovnat k bublinam s pismenami, ktore su prepojene navzajom medzi sebou, asi kazdy s kazdym, pri slove JIM sa najprv odignoruju A-J bubliny a vyberie J, v druhom kroku odignoruje A-I bubliny a vyberie I, atd. System zvycajne sam rozhodne, ktory index pouzit a ci pri vyhladani ho vobec pouzije, lebo inak spravi **FULL TABLE SCAN**, cize prechadza hodnoty riadok po riadku, pri velmi velkom pocte hodnot by to ale bolo pomale, tak pouzije indexy. Indexy sa oplati pouzivat na tie stlpce(udaje), ktore casto pouzivam v JOIN, WHERE, GROUP BY, ORDER BY, hlavne teda pri tych porovnaniach (WHERE) to pracu urychli.

Nevyhodou indexov moze byt, ze zaberaju urcite miesto (pre kazdy riadok) a pri velkom pocte zaznamov to moze byt na skodu. Preto treba pouzitie zvazit, pri malom pocte zaznamov moze byt rychlejsi FULL TABLE SCAN. Pri pouziti indexov su vsetky INSERT a UPDATE pomalsie, dalsia nevyhoda.

**UNIQUE**: pracuje ako index, ale este navyse kontroluje unikatnost hodnoty(napr. email, nemoze byt viacero rovnakych mailov), ci uz nie je niekde v tabulke.

**EXPLAIN** SELECT \* FROM directors WHERE first\_name='Cool1'

//pri tomto prikaze(este bez EXPLAIN) som vyskusal vyhladanie s/bez indexov, pricom s indexami to bolo aspon o 100 krat rychlejsie vykonane, **EXPLAIN** nam sluzi ako nejaka informacia, ktora vypise ze cez kolko zaznamov query preslo

**Prikazovy riadok** spustenie MYSQL:

cd D:\Programovanie\MAMP\bin\mysql\bin

mysql.exe -u root -p -h localhost //heslo root, alebo co nastavim

dalsie prikazy uz v MySQL:

SHOW databases;

USE movies; //pouzitie tabulky, po tomto prikaze mozem pisat sql prikazy z movies

SHOW TABLES;

//praca cez konzolu je rychlejsia, bezpecnejsia ale zaroven menej prehladna

delimiter //

CREATE **PROCEDURE** dorepeat(p1 INT)

BEGIN

SET @x = 0;

**REPEAT**

SET @x = @x + 1;

INSERT INTO `directors` (`first\_name`, `last\_name`, `country`, `birthdate`)

VALUES (CONCAT('Cool', @x), CONCAT('Dude', @x), 'Spain', now());

**UNTIL** @x > p1 END REPEAT;

END//

//delimiter je oddelovac a pouziva znak “//” rovnaky ako ja na komenty, cely ten kod je vytvorena procedura(funkcia), ktora nam bude nejake prikazy medzi REPEAT a UNTIL opakovat tolko krat, kolko zadame do funkcie pri volani, volanie tejto procedury potom vyzera takto: call dorepeat(500);

Ked sa dava databaza na realnu stranku, tak sa pouziva databaza typu Mariadb, pretoze tato je uplne zdarma, kod mariadb a mysql je skoro uplne rovnaka, ale vyhadzuje to niekedy chyby. Defaultne sa databazy nastavuju na engine InnoDB. Kazdy engine uklada subor inym sposobom a kazdy je dobry na nieco ine. Vieme ich zistit cez prikazovy riadok: **SHOW ENGINES**; alebo aj verziu MySQL **SELECT VERSION()**; Najpouzivanejsie enginy su **MyIsam** a **InnoDB**. InnoDB podporuje atomicky update, MyIsam nie, je to update, ktory ked v behu prerusim, tak sa uplne zrusi, nezostane nic obnovene. InnoDB podporuje vykonavanie viacej queries naraz, pretoze pri vykonavani sa zablokuje len riadok tabulky, pri MyIsam sa zablokuje cela tabulka, tak je mozna len vykonanie jednej query. MyIsam sa pouziva pri FullText indexovani pokial ide o starsie SQL(pri starsej sql nebolo podporovane FULLTEXT indexovanie na InnoDB).

**Transakcia** sa pouziva napr pri prevodoch v bankach, kde vykonavanie queries musi byt spojite,aj ked zlyha databaza, cize ak dojde k vykonaniu jedneho prikazu a naslednemu zlyhaniu, zmena nemoze nastat, len v pripade ze sa vykonaju oba prikazy.

**CSV** je engine, ktora klasicky oddeluje dvojbodkami alebo bodkociarkou jednotlive slova, je to dobre na export suborov, CSV je kompatibilne napr s excelom, textovym suborom.

**Memory** engine uklada hodnoty do RAMKY, cize pouzivam vtedy ked mi nebude vadit strata dat po vypnuti pocitaca.

Pri **BlackHole** engine sa data stratia okamzite, dobre napr pri zalohovani pocitaca, kde mi zalohuje vsetky data, ale tabulky nastavene na blackhole si pri zalohovani stratia. Pokial ak ale budem chciet pridat nove data do tejto blackhole tabulky, nevyhodi mi chybu, ale tieto data mi proste zahodi.

Archive je pomale indexovanie, urcene len na archivacne ucely.

**FULLTEXT** indexovanie je klasicke vyhladavanie napr na stranke, kde mi vyhadzuje aj slova, ktore sa podobaju vyhladavanemu textu, nemusia uplne sediet a nasledne im dava aj vahu, ktore sa podla neho najviac hodi pre nase vyhladavanie. Niektore FULLTEXT podporuju aj slovnik a vyhladavaju aj mnozne cislo daneho slova, minuly cas slovesa, atd. Ak pouzijem LIKE pomocou WILDCARD(%) tak mi bude vyhladavat FULL TEXT SCAN a odignoruje jednotlive indexy, cize to bude pomale.

SELECT id, title, summary FROM movies WHERE **MATCH** (summary) **AGAINST** ('man finds love')

//kombinacia **MATCH** a **AGAINST** sa pouziva pre **FULLTEXT** vyhladavanie, treba ale nastaviť dany stlpec(summary) na index FULLTEXT. InnoDB podporuje FULLTEXT len pre jeden stlpec, nevedel som teda pridat aj title do MATCH-u

SELECT \* FROM movies WHERE **SOUNDEX**(title) = **SOUNDEX**('Htt Fzz')

//najde mi udaje, ktore zneju(v anglictine) ako Hot Fuzz, napr htt fzz

SELECT id, title, summary, **MATCH (summary) AGAINST ('man finds love') AS score** FROM movies WHERE MATCH (summary) AGAINST ('man finds love')

//ak dam match do selectu, tak mi vypise score zhody, podla coho usporiadava dolezitost vysledku

SELECT id, title, summary,

MATCH (summary) AGAINST ('hero >woman' **IN BOOLEAN MODE**) AS score

FROM movies

ORDER BY score DESC

//**BOOLEAN** mod na dovoluje urcovat dolezitost vyhladavania, v tomto pripade ma ‘woman’ vacsiu prioritu ako ‘hero’, priorita sa zvysi o 1(v predoslom priklade sme pocitali score)

//pri boolean mode viem pridavat modifikatory(znamienka) aj wildcardy do vyhladavania, aj vyclenit slova, ktore sa nesmu vyskytovat vo vysledku

Dalsie moznosti(https://dev.mysql.com/doc/refman/5.6/en/fulltext-boolean.html):

HERO SA MUSI VYSKYTOVAT

love\* +hero //hviezdicka znamena, ze slovo pre slovo love bude hladat aj mnozne cisla,atd

SEX A HERO SA NESMU VYSKYTOVAT

-hero -sex

SELECT id, title, summary,

MATCH (summary) AGAINST ('man' **WITH QUERY EXPANSION**) AS score

FROM movies

ORDER BY score DESC

//**WITH QUERY EXPANSION** funguje tak, ze najprv najde vysledky s danym slovom, potom dalej hlada dalsie vysledky, ktore suvisia s celym textom, ak sa v prvom vysledku napr. najde slovo young, tak dalej bude hladat vysledky so slovom young a aj tym priradi nejaku prioritu

**Relacie:**

Pokial by som chcel zmenit **InnoDB** na **MyIsam**, tak by to vypisalo chybu kvoli relaciam(prepojom), ktore asi MyIsam nepodporuje, preto by ich trebalo najprv zmazat a az potom zmenit tabulku. Kvoli tymto **prepojom neni mozne zmazat** niektore udaje, pokial su prepojene s inou tabulkou, tak nevedel by co ma spravit s udajmi z druhej tabulky. Riesenie je take, ze zmenim v prepojeniach(Cizi klice) dany stlpec na **SET NULL** pre akcie ON UPDATE a ON DELETE a umoznim pre danu premennu hodnotu NULL. V tomto pripade, ak zmazem nejaky udaj z jednej tabulky, do prepojenej tabulky mi vyhodi NULL na dane hodnoty.

Miesto **SET NULL** ak pouzijem **CASCADE**, zmeny robi automaticky aj v druhej tabulke tiez mazania, hodnoty sa teda prepojene menia. Pre **CASCADE** aby to bolo plne funkcne, treba spat nastavit neumoznenie NULL pre dany prepojeny stlpec.

Ak chcem **zmazat** niektoru tabulku, ktora je prepojena s inou, **nebude** to **dovolene** za relacie. Musim to spravit nasledujuco, aby sa to dalo zmazat:

**SET FOREIGN\_KEY\_CHECKS = 0;**

DROP TABLE IF EXISTS genres;

**SET FOREIGN\_KEY\_CHECKS = 1;**

// trik je v tom, ze docasne vypnem prepojenie klucov a nasledne po zmazani ich znova nastavim

Niektore veci v MySQL su dovolene, napr delenie nulou, alebo ak nastavim pre niektory stlpec z viacerych ze nemoze byt nulovy a pridam hodnoty len do ostatnych hodnot, automaticky mu hodi defaultnu hodnotu, nevypise error. Tento error nevypise preto, lebo to ma nastavene MySQL v defaultnych nastaveniach, ktore my mozeme zmenit prikazom:

SET GLOBAL **SQL\_MODE** = 'TRADITIONAL'

Tieto nastavenia sa nasledne zobrazia v subore my.cnf .Pri errore “My SQL server has gone away” treba zmenit pocet dovolenych dat na jeden prikaz, ktore najdeme v cnf subore pod “max\_allowed\_packet”(pre XAMP, MAMP?).

# **SQLite**