Training: MySQL Grundlagen (Hannover)

Agenda

1. Technical Background
    - Technical Structure
    - Process of Queries
2. CONSTRAINTS
    - Foreign Key Constraints - Example

1. Working with database objects
    - Working with databases
    - Working with tables
    - Teststellung - Feld verkleinern
2. SELECT
    - Select Beispiele
    - Select Beispiele mit Like
    - SELECT ORDER BY
    - Unterschiede einfache und doppelte Hochkommas bei Oracle/MySQL
3. INSERT/UPDATE
    - Praktisches Beispiel und erweitertes insert - INSERT
    - Praktisches Beispiel - Update
4. DELETE
    - Einfaches Delete Beispiel
    - Delete mit Transaktion
5. JOINS
    - Basics of Joins
    - Working with LEFT JOIN
    - Join examples
6. GROUP BY
    - Simple Group By Example
    - Join and group - example
7. Datentypen
    - Integer - INT - Datentypen
8. Basics
    - Connection to DB + exit
    - mysql-client
    - Charset-Collations
    - server system variablen abfragen
9. Storage Engines

- Which engine is used

10. Working with the data modelling language (DML's)

    - Working with INSERT

11. Tipps & Tricks / Do Not

    - SQL-Query im Query Tab (MySQL Workbench) direkt ausführen
    - Dump/SQL-File einspielen auf der Kommandozeile - Windows
    - Möglichst keine Funktion in where (spalte) verwenden
    - Export Partial Columns in MYSQL with INTO OUTFILE

12. Tools

    - HeidiSQL Portable - works for windows

13. References

    - Examples Left Join
    - Notes on Specific MySQL Knowledge
    - Many Sakila Example Queries
    - Sakila Testdatenbank
    - SQLplus Oracle Commandline - Client
    - Welche Datentypen (aus anderen Datenbank-Systemen z.B. Oracle) in MySQL verwenden
    - MySQL Performance - pdf
    - Helpful Examples

14. Extra (Optional)

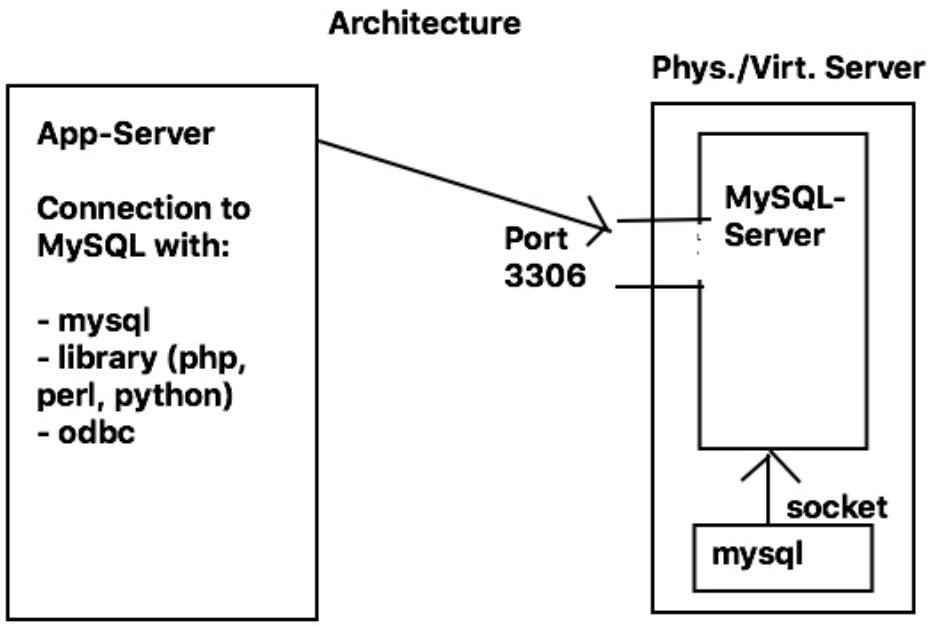
    - Explain
    - Indizes
    - Schnellster Import von Daten mit csv

15. Übungen

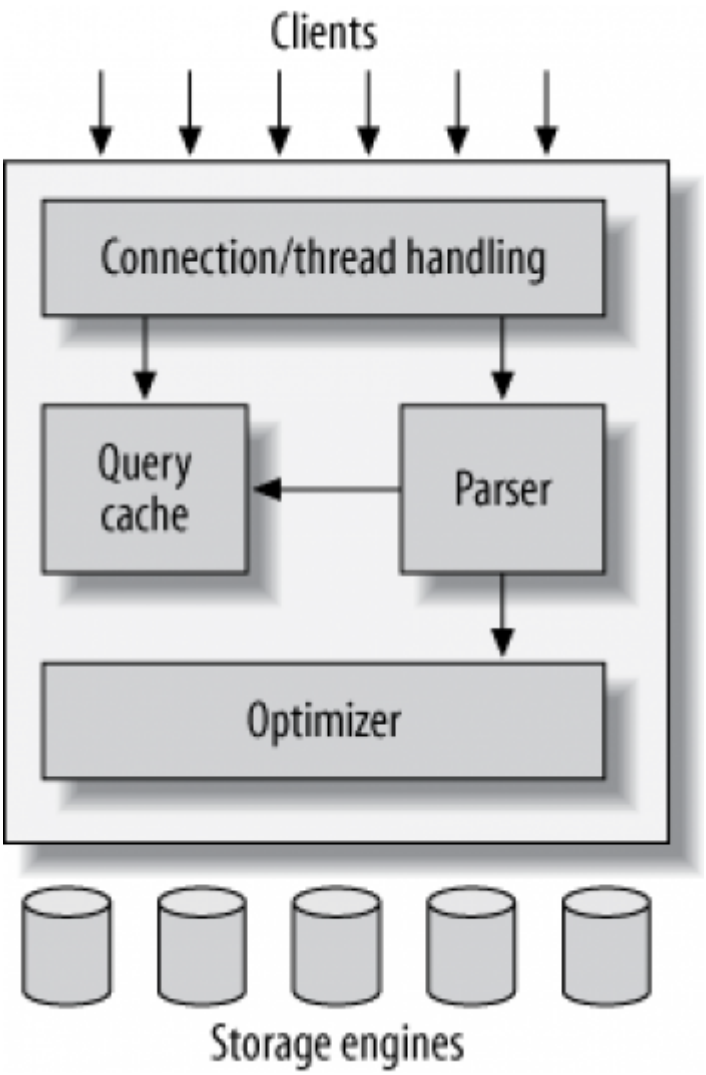
    - Übung Update/Insert

Technical Background

Technical Structure



Process of Queries



CONSTRAINTS

Foreign Key Constraints - Example

```
## Walkthrough
create filmcopy as select * from film:
set foreign_key_checks = 0;
trunate film
set foreign_key_checks = 1
alter table actorcopy add constraint (fk_language_id) references language
(language_id)

## Test it
## language.language_id needs to have an index
delete from language where language_id = 1
```

Working with database objects

Working with databases

Explanations

```
## open a connection to the mysql-server by entering
mysql
## then you will get
mysql>
```

```
## Comments within mysql-client
## three - in a row
---
```

Create database

```
create database training
create schema training2
```

Show databases

```
mysql
mysql>
;; from here i leave out mysql>
;; so you can easily copy & paste the lines hereafter
show databases

--
-- or
--

show schemas

--
-- or by using information_schema
--

select * from information_schema.schemata;
```

Working with tables

Show tables

```
## within mysql>
## so on the command-line enter:
## mysql (as root)
USE sakila
SHOW TABLES

-- or --

select * from information_schema.TABLES
```

Create table

```
-- only if you want to create table in a completely new database
create schema training;
USE training
CREATE TABLE people (id INT NOT NULL AUTO_INCREMENT, name VARCHAR(20), PRIMARY
KEY(id));
```

Find out the structure of the table

```
## you have to connect to db first with
## mysql
## within mysql>
DESCRIBE people
SHOW CREATE TABLE people
-- or : if you want to know more --
SELECT * from INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='people' AND
TABLE_SCHEMA='training' \G
```

Show indexes

```
SHOW INDEX FROM actor
SHOW INDEXES FROM ACTOR
```

Change table (Add field)

```
--- We want to add a field before name
--- IMPORTANT: BEFORE does not exist
ALTER TABLE people ADD first_name VARCHAR(10) AFTER id;

ALTER TABLE schulungen ADD seats TINYINT unsigned DEFAULT 1, ADD price DECIMAL(6,2);
ALTER TABLE schulungen ADD (room TINYINT unsigned DEFAULT 1,  discount DECIMAL(6,2));
```

Modify a field in table (Change property)

```
ALTER TABLE people
    MODIFY COLUMN first_name VARCHAR(20);
```

Drop a field from the table

```
ALTER TABLE people ADD middle_name VARCHAR(25) BEFORE name;
DESCRIBE people;
ALTER TABLE people DROP COLUMN middle_name;
```

```
## More Examples
--
ALTER TABLE actor ADD in_rente BOOLEAN default true
INSERT INTO actor (first_name,last_name,in_rente) values ('Jochen','Metzger',false)
-- Wieder loswerden
ALTER TABLE actor DROP in_rente;
## add and drop in once command
ALTER TABLE actor ADD in_rente2 BOOLEAN default true, DROP in_rente;
```

Deleting table data (truncate)

```
USE sakila
-- Create table based on other table
CREATE TABLE actorcopy as SELECT * FROM actor;
-- Fields ?
SELECT * FROM actorcopy;
-- Empty it
TRUNCATE TABLE actorcopy;
-- Emptry ?
SELECT COUNT(*) FROM actorcopy;
```

Delete table data (with delete)

Explanation

- Do not use delete when you want to use data of complete table
  - truncate is quicker in this case.
- DELETE FROM ... WHERE ... does a SELECT first

Example

```
USE sakila
CREATE TABLE actorbackup AS SELECT * FROM actor;
SELECT COUNT(*) FROM actorbackup;
DELETE FROM actorbackup WHERE actor_id > 100;
SELECT COUNT(*) FROM actorbackup;
```

Delete complete table

```
USE sakila
DROP TABLE actorbackup;
```

Teststellung - Feld verkleinern

```
ALTER TABLE actor ADD filme tinyint unsigned default 255;
-- Does not work
-- UPDATE actor SET filme = 256  WHERE actor_id = 1;
-- ALTER TABLE actor MODIFY filme smallint unsigned default 256;
INSERT INTO actor (first_name,last_name) values ('Gaucho','Poncho');
INSERT INTO actor (first_name,last_name,filme) values ('Gauchina','Ponchina',32000);

-- Does not work
ALTER TABLE actor MODIFY filme tinyint unsigned;
```

SELECT

Select Beispiele

Einfaches Beispiel (bestimmte Felder)

```
## Zeige alle diese felder aus dieser Tabelle
SELECT <welche_feld_mit_komma_getrennt> FROM <welche tabelle>

-- bitte datenbank sakila verwenden
use sakila
-- zeige actor_id, last_name
SELECT actor_id, last_name FROM actor
```

Einfaches Beispiel (alle Felder)

```

-- bitte datenbank sakila verwenden
use sakila
-- zeige actor_id, last_name
SELECT * FROM actor
```

Einfaches Beispiel mit Bedingung

```
-- SYNTAX
use sakila;
-- SELECT <welche_felder> FROM <welche_tabelle> WHERE
<wo_welches_feld_welchen_wert_hat>
-- Beispiel 1
SELECT actor_id, last_name FROM actor where actor_id = 5;
-- Beispiel 2
SELECT * FROM actor where actor_id = 5;
```

Einfache Bedingung mit Bereich(en)

```
-- SYNTAX
use sakila;
-- SELECT <welche_felder> FROM <welche_tabelle> WHERE
<wo_welches_feld_welchen_wert_hat>
-- Beispiel 1
SELECT actor_id, last_name FROM actor where actor_id > 8;
-- Beispiel 2
SELECT * FROM actor where actor_id > 8;
-- Beispiel 3
SELECT * FROM actor where actor_id > 8 and actor_id < 50;
```

Zwei Bereiche abfragen

```
## Brackets are not necessary, works the same
select * from actor where (actor_id >= 8  and actor_id <=50) or (actor_id >= 100 and
```

```
actor_id <= 150)
```

Select Beispiele mit Like

Hintergrund Platzhalter '%' und '_'

- https://elearn.inf.tu-dresden.de/sqlkurs/lektion01/01_07_03_einschr_like.html

Name fängt mit J an

```
## Alle Datensätze, die mit J anfangen
select first_name,last_name from actor where last_name like 'j%'
## mit ausgabe zusätzlichem String
select first_name,last_name,' ist der/die Beste' as bewertung from actor where
last_name like 'j%'
```

Name hört mit n auf

```
select first_name,last_name from actor where last_name like '%n'
```

Name beinhaltet 'q'

```
select first_name,last_name from actor where last_name like '%q%'
```

Platzhalter für genau ein Zeichen (Linux/Windows -> ?)

```
## Alle Zeilen mit Last name McQ, dann genau einem beliebigen Zeichen und dann 'een'
SELECT * FROM sakila.actor where last_name like 'McQ_een'
```

SELECT ORDER BY

Syntax

```
-- Variante 1: (ohne where)
-- SELECT * FROM <welche_tabelle> ORDER BY <welches_feld>


-- Variante 2: (mit where)
-- SELECT * FROM <welche_tabelle> WHERE <welche_bedingung> ORDER BY <welches_feld>
```

Beispiel ohne where

```
## feld aufsteigend
SELECT last_name,first_name,actor_id FROM sakila.actor ORDER BY last_name

## feld absteigend
SELECT last_name,first_name,actor_id FROM sakila.actor ORDER BY last_name desc

## erstes feld aufsteigend, zweites feld absteigend
SELECT last_name,first_name,actor_id FROM sakila.actor ORDER BY last_name,first_name
DESC
```

Beispiel mit where

```
SELECT last_name,first_name,actor_id FROM sakila.actor WHERE last_name like 'J%' ORDER
BY last_name ASC,first_name DESC
```

Unterschiede einfache und doppelte Hochkommas bei Oracle/MySQL

MySQL

```
Hochkommas werden nur bei der Abfrage des Wertes verwendet, z.B

select * from actor where last_name like 'A%'
## das ist das gleiche wie:
select * from actor where last_name like "A%"
```

MySQL - Wann nehme ich einfache, wann doppelte ?

```
## z.B. wenn ich ein Hochkomma in der Abfrage brauche, z.B
## damit besser lesbar für Admin/Entwickler
select * from actor where last_name like "O'Connor"
## Alternativ geht auch:
## Verfahren: Escapen
select * from actor where last_name like 'O\'Connor'
```

Oracle:

```
## Für Feldname und Ausgabe Feldname (Alias) Doppelte hochkommas.
## z.B
## Hier immer das doppelte Hochkomma, weil es um den Identifier/Bezeichner geht
SELECT  'Hello, world!'    AS "My Greeting"

### Für String und Date - Werte einfache Hochkommas
select * from actor where last_name like 'A%'
```

INSERT/UPDATE

Praktisches Beispiel und erweitertes insert - INSERT

Beispiel

```
INSERT INTO actor (first_name,last_name) values ('Joe','Manchos');
```

Erweitertes Insert

```
## Mehrere Wertepaare einfügen - geht schneller als einzelne Inserts
INSERT INTO actor (first_name,last_name) values ('Joe','Metzgeros'),
('Hans','Mustermann');
```

Referenz

- https://www.w3schools.com/sql/sql_insert.asp

Praktisches Beispiel - Update

Einfaches Beispiel ein Datensatz ändern

```
## Variante 1
UPDATE actor SET last_name = 'GUINESSA' WHERE actor_id = 1
## Variante 2 - 2 Felder ändern
UPDATE actor SET first_name='PENELOPEZ',last_name = 'GUINESS' WHERE actor_id = 1
```

Referenz

- https://www.w3schools.com/sql/sql_update.asp

DELETE

Einfaches Delete Beispiel

Example

```
DELETE FROM actor WHERE id = 200
```

Reference

- https://www.mysqltutorial.org/mysql-delete-statement.aspx

Delete mit Transaktion

Beispiel

```
Variante 1: Andere sehen es erst nach commit (andere Sessions/Verbindungen)

BEGIN;
DELETE FROM actor where actor_id = 200;
COMMIT;

Variante 2: Aktion mir nicht, ich rolle sie zurück, mache sie rückgängig
BEGIN;
DELETE FROM actor where actor_id = 200;
ROLLBOCK;
```

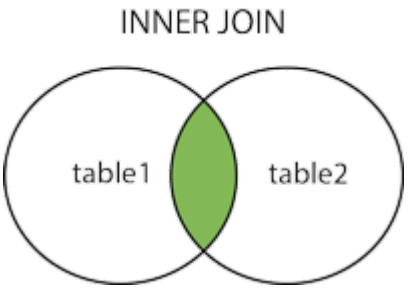
JOINS

Basics of Joins

What is a JOIN for ?

- combines rows from two or more tables
- based on a related column between them.

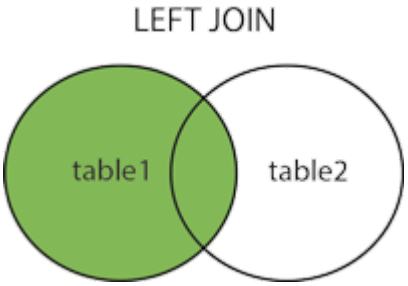
MySQL (Inner) Join



MySQL (Inner) Join (explained)

- Inner Join and Join are the same
- Returns records that have matching values in both tables
- Inner Join, Cross Join and Join
    - are the same in MySQL

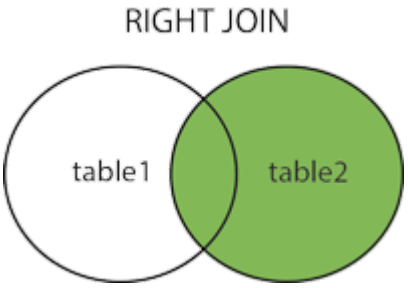
MySQL Left Join



MySQL Left (outer) Join (explained)

- Return all records from the left table
- *AND* the matched records from the right table
- The result is NULL on the right side
    - if there are no matched columns on the right
- Left Join and Left Outer Join are the same

MySQL Right Join



MySQL Right Join (explained)

- Return all records from the right table
  - *AND* the matched records from the left table
- Right Join and Right Outer Join are the same

MySQL Straight Join

- MySQL (inner) Join and Straight Join are the same
- **Difference:**
  - The left column is always read first
- **Downside:**
  - Bad optimization through mysql (query optimizer)
- **Recommendation:**
  - Avoid straight join if possible
  - use join instead

Type of Joins

- [inner] join
  - **inner join** and **join** are the same
- left [outer] join
- right [outer] join
- full [outer] join
- straight join < equals > join
- cross join = join (in mysql)
- natural join <= equals => join (but syntax is different)

In Detail: [INNER] JOIN

- Return rows when there
  - is a match in both tables
- Example

```
SELECT actor.first_name, actor.last_name, film.title
FROM film_actor
INNER JOIN actor ON film_actor.actor_id = actor.actor_id
INNER JOIN film ON film_actor.film_id = film.film_id;
```

In Detail: Joining without JOIN - Keyword ====

- Explanation: Will have the same query execution plan as [INNER] JOIN

```
SELECT actor.first_name, actor.last_name, film.title
FROM film_actor,actor,film
where film_actor.actor_id = actor.actor_id
and film_actor.film_id = film.film_id;
```

In Detail: Left Join

- Return all rows from the left side
  - even if there is not result on the right side
- Example

```
SELECT
c.customer_id,
c.first_name,
c.last_name,
a.actor_id,
a.first_name,
a.last_name
FROM customer c
LEFT JOIN actor a
ON c.last_name = a.last_name
ORDER BY c.last_name;
```

In Detail: Right Join

- Return all rows from the right side
  - even if there are no results on the left side
- Example

```
SELECT
c.customer_id,
c.first_name,
c.last_name,
a.actor_id,
a.first_name,
a.last_name
FROM customer c
RIGHT JOIN actor a
ON c.last_name = a.last_name
ORDER BY a.last_name;
```

In Detail: Having

- Simple: WHERE for GroupBy (because where does not work here)
- Example

```
SELECT last_name, COUNT(*)
FROM sakila.actor
GROUP BY last_name
HAVING count(last_name) > 2
```

Internal (type of joins) - NLJ

- NLJ - (Nested Loop Join)

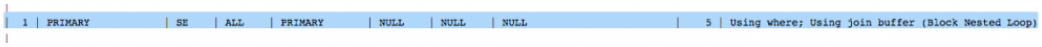
```
for each row in t1 matching range {
for each row in t2 matching reference key {
for each row in t3 {
  if row satisfies join conditions, send to client
}
}
}
```

Internal (type of joins) - BNL

- BNL - (Block Nested Loop)
  - in explain: -> using join buffer
  - columns of interest to a join are stored in join buffer
    - --> not whole rows.
  - join_buffer_size system variable
    - -> determines the size of each join buffer used to process a query.
- https://dev.mysql.com/doc/refman/5.7/en/nested-loop-joins.html

BNL - Who can I see, if it is used ?

- Can be seen in explain




```
explain SELECT a.* FROM actor a  INNER JOIN actor b where a.actor_id > 20 and
b.actor_id < 20
```

```
When using a Block Nested-Loop Join, MySQL will, instead of automatically joining t2,
insert as many rows from t1 that it can into a join buffer
and then scan the appropriate range of t2 once,
matching each record in t2 to the join buffer.
From here, each matched row is then sent to the next join,
which, as previously discussed, may be another table,
t3, or, if t2 is the last table in the query,
the rows may be sent to the network.
```

BNL's - Refs:

- https://www.burnison.ca/notes/fun-mysql-fact-of-the-day-block-nested-loop-joins

Working with LEFT JOIN

General

- Show all entries from the left table and only from the right if available
- Examples are based on sakila database.

Example - new language / not in language table

```
SELECT @@foreign_key_checks;
SET FOREIGN_KEY_CHECKS=0
UPDATE film  SET language_id = 99 WHERE film_id >= 800 and film_id <= 899
SELECT f.film_id,f.title,f.description,l.name FROM film f LEFT JOIN language l ON
f.language_id = l.language_id;
SET FOREIGN_KEY_CHECKS=1
```

Example

```
use sakila
SELECT
    c.customer_id,
    c.first_name,
    c.last_name,
    a.actor_id,
    a.first_name,
    a.last_name
FROM customer c
LEFT JOIN actor a
ON c.last_name = a.last_name
ORDER BY c.last_name;
```

Join examples

```
## Work - step by step.
SELECT * FROM film f;
SELECT f.title FROM film f;
SELECT f.language_id FROM film f JOIN language l ON f.language_id = l.language_id;
-- aus film-tabelle alle felder - f.*
SELECT f.*,l.name FROM film f JOIN language l ON f.language_id = l.language_id;
SELECT f.film_id,f.title,l.name,f.description FROM film f JOIN language l ON
f.language_id = l.language_id;
```

GROUP BY

Simple Group By Example

```
## Variante 1
SELECT last_name,COUNT(last_name) as cnt FROM actor GROUP BY last_name

## Variante 2: ohne Group - akroyd zählen -> geht nur bei einem Namen
SELECT last_name,COUNT(last_name) as cnt FROM actor WHERE last_name = 'AKROYD'

## Das ist falsch - weil mehrere Namen, Ausgabe nur eine Zeile
SELECT last_name,COUNT(last_name) as cnt FROM actor WHERE last_name = 'AKROYD' or
last_name = 'ALLEN'

## Variante 2a (Erst Daten holen - 6 Datensätze, dann aggregieren (group by)
## Für grosse Datenmengen besser !
SELECT last_name,COUNT(last_name) as cnt FROM actor WHERE last_name = 'AKROYD' or
last_name = 'ALLEN' GROUP BY last_name
## Variante 2b (Alle Daten holen - 200 Datensätze, dann alles aggregieren)
SELECT last_name,COUNT(last_name) as cnt FROM actor GROUP BY last_name HAVING
last_name = 'AKROYD' or last_name = 'ALLEN'
```

Join and group - example

Datentypen

Integer - INT - Datentypen

Reference

- https://dev.mysql.com/doc/refman/8.0/en/integer-types.html

Basics

Connection to DB + exit

General Explanation

- Step 1: connection to db
- Step 2: using specific database
- 1. ▪ 2. can be done in one step

Connection as Root (without using -u/--user and db)

```
## If you are root and are connecting locally (socket), you do not need to enter a
passwort on root
## Why ?
mysql> use mysql
mysql> select user,host,plugin from user where user = 'root' and host = 'localhost';
```

```
root@mysql-server:~#whoami
root
## this work, because we are connecting locally
## by default mysql uses the user we are logged in with
mysql
```

Connection with credentials

```
mysql -uroot -p
## passwort auf der Kommandozeile eingeben
```

Connection to remote mysql - server

```
mysql -u root -p -h 10.10.9.117
```

mysql-client

Basics

```
mysql
mysql>

## Wie kommen wie raus ?
exit;
```

Delimiter

```
Normalerweise ";"

Ist zum Trennen von Befehlen
```

Use user and password automatically

```
nano /root/.my.cnf
## BE CAREFUL EVERYBODY CAN LOGIN AS ROOT TO MYSQL NOW
## in there
[mysql]
user=root
password=root-password-on-your-system
```

Charset-Collations

server system variablen abfragen

```
mysql> show session variables like '%hostname%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| hostname      | trn01 |
+---------------+-------+
1 row in set (0.00 sec)

mysql> select @@hostname;
+------------+
| @@hostname |
+------------+
| trn01      |
+------------+
1 row in set (0.00 sec)
```

Storage Engines

Which engine is used

```
show variables like '%default%';
```

Working with the data modelling language (DML's)

Working with INSERT

```
## Always use the field-names
INSERT INTO actor (first_name,last_name) values ('John','Smith');

## Extended inserts are quick (better than single inserts)
INSERT INTO actor (first_name,last_name) values ('John','Peters'),
('Mandy','Johnsson');
```

Tipps & Tricks / Do Not

SQL-Query im Query Tab (MySQL Workbench) direkt ausführen

```
## Statt Blitz-Symbol anzuklicken, kann man einfach die Tastenkombination
## STRG + ENTER - verwenden (dann wir es direkt ausgeführt)
```

Dump/SQL-File einspielen auf der Kommandozeile - Windows

```
mysql -u root -p < C:\Users\Admin\Downloads\test.sql
```

Möglichst keine Funktion in where (spalte) verwenden

```
## Bad for performance

## Bad


## Good
```

Export Partial Columns in MYSQL with INTO OUTFILE

```
-- zeig mir das Datenverzeichnis
SELECT @@datadir;
-- der secure-path - nur dort darf hin exportiert werden
show variables like '%secure%';

SELECT actor_id,last_name
INTO OUTFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\result.txt'
FROM actor;
```

Tools

HeidiSQL Portable - works for windows

- https://www.heidisql.com/download.php?download=portable-64

References

Examples Left Join

- https://www.quackit.com/mysql/examples/mysql_left_join.cfm

Notes on Specific MySQL Knowledge

- https://www.burnison.ca/notes

Many Sakila Example Queries

- https://github.com/ashok-bidani/MySQL-Sakila-queries-and-joins

Sakila Testdatenbank

- https://dev.mysql.com/doc/index-other.html

SQLplus Oracle Commandline - Client

- https://docs.oracle.com/cd/B10501_01/server.920/a90842/qstart.htm

Welche Datentypen (aus anderen Datenbank-Systemen z.B. Oracle) in MySQL verwenden

- https://dev.mysql.com/doc/refman/8.0/en/other-vendor-data-types.html

MySQL Performance - pdf

- https://schulung.t3isp.de/documents/pdfs/mysql/mysql-performance.pdf

Helpful Examples

- https://www.quackit.com/mysql/examples/mysql_group_by_clause.cfm

Extra (Optional)

Explain

```
explain format=json  SELECT a.first_name, a.last_name, fa.film_id FROM film_actor2 fa
INNER JOIN actor2 a ON fa.actor_id = a.actor_id
```

What does type say ?

- https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain-join-types

Indizes

Avoid ALL

- is the worst type : TABLE SCAN (Need to go through all rows)

```
mysql> create table actor4 as select * from actor;
mysql> explain select * from actor4 where actor_id > 10;
+----+-------------+--------+------------+------+---------------+------+---------+----
--+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref
| rows | filtered | Extra       |
+----+-------------+--------+------------+------+---------------+------+---------+----
--+------+----------+-------------+
|  1 | SIMPLE      | actor4 | NULL       | ALL  | NULL          | NULL | NULL    |
NULL |  200 |    33.33 | Using where |
+----+-------------+--------+------------+------+---------------+------+---------+----
--+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

Cover Index.

- We can get all the necessary information from the index (no acces of filesystem necessary)

```
drop table if exists actor2;
create table actor2 as select * from actor;
create index idx_actor2_last_name on actor2 (last_name);

## using index
## <- indicates that a cover index is used

mysql> explain select last_name from actor2 where last_name like 'B%';
+----+-------------+--------+------------+-------+-----------------------+-------------
---------+---------+------+------+----------+--------------------------+
| id | select_type | table  | partitions | type  | possible_keys         | key
| key_len | ref  | rows | filtered | Extra                    |
+----+-------------+--------+------------+-------+-----------------------+-------------
---------+---------+------+------+----------+--------------------------+
|  1 | SIMPLE      | actor2 | NULL       | range | idx_actor2_last_name |
idx_actor2_last_name | 182     | NULL |   22 |   100.00 | Using where; Using index |
+----+-------------+--------+------------+-------+-----------------------+-------------
---------+---------+------+------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

Creating a primary index

```
create index primary key on actor2 (actor_id)
explain select actor_id from actor2 where actor_id > 2
```

Using an index for last_name

```
drop table if exists actor2;
create table actor2 as select * from actor;
create index idx_actor2_last_name on actor2 (last_name);
explain select * from actor2 where last_name like 'B%';
+----+-------------+--------+------------+-------+-----------------------+-------------
---------+---------+------+------+----------+-----------------------+
| id | select_type | table  | partitions | type  | possible_keys         | key
| key_len | ref  | rows | filtered | Extra                 |
+----+-------------+--------+------------+-------+-----------------------+-------------
---------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | actor2 | NULL       | range | idx_actor2_last_name |
idx_actor2_last_name | 182     | NULL |   22 |   100.00 | Using index condition |
+----+-------------+--------+------------+-------+-----------------------+-------------
---------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

Never use a function in where

Why ?

```
Step 1: MySQL needs to retrieve every row
Step 2: run function
--> so, no index can be used
```

Example

```
drop table if exists actor2;
create table actor2 as select * from actor;
create index idx_actor2_last_name on actor2 (last_name);
explain select * from actor2 where last_name like
concat(substring(first_name,1,1),'%');
```

Index is always read from left to right

```
## so the index cannot be used if we ask for last_name
drop table if exists actor2;
create table actor2 as select * from actor;
create index idx_actor2_first_name_last_name on actor2 (first_name,last_name);
explain select * from actor2 where last_name like 'B%';
##
explain select * from actor2 where first_name like 'B%';
```

Schnellster Import von Daten mit csv

Example

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
  FIELDS TERMINATED BY ','  LINES STARTING BY 'xxx';

LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

General/Ref

- Is the quickest way
- Performance Ref: https://jynus.com/dbahire/testing-the-fastest-way-to-import-a-table-into-mysql-and-some-interesting-5-7-performance-results/

Übungen

Übung Update/Insert

```
1. In einer Anweisung alle Datensätze ändern in denen der erste Name JUDY und der
Nachname DEAN ist -> dort Vorname in JAMES ändern.

UPDATE actor SET first_name = 'JAMES' where last_name='DEAN' and first_name='JUDY';


2. In einem SQL-Statement 2 neue Datensätze einfügnen. 1. Mann, Josef  2. Mannheim,
Martha

INSERT INTO actor (first_name, last_name) values ('Mann','Josef'),
('Mannheim','Martha');
```