Training: MySQL Grundlagen/Performance (Hannover)

Agenda

1. Technical Background
    - Technical Structure
    - Process of Queries
2. Beispieldaten / Testserver
    - Sakila
    - Spendenliste
    - Server Vagrant
3. JOINS
    - Basics of Joins
    - Working with LEFT JOIN
    - Join examples
4. GROUP BY
    - Simple Group By Example
    - Join and group - example
5. CONSTRAINTS
    - Foreign Key Constraints - Example
6. UPDATE
    - Sophisticated Update
7. JSON
    - Short Introduction working JSON
8. Performance
    - * vs. specific field in field list - select
    - Möglichst keine Funktion in where (spalte) verwenden
    - SQL-Rewrite Pager- Subselect
9. Analyzing Slow Queries / Indexes
    - Find indexes
    - Create Index/Delete/Drop Index
    - Indizes
    - Explain
    - profiling-get-time-for-execution-of.query
    - Kein function in where verwenden
    - Optimizer-hints (and why you should not use them)
    - Query-Plans aka Explains
    - Query Pläne und die Key-Länge
    - Index und Likes
    - Index und Joins
    - Find out cardinality without index
    - Index and Functions

10. Tools
    - Percona Toolkit
    - pt-query-digest - analyze slow logs
    - pt-online-schema-change howto
    - Example sys-schema and Reference
11. Storage Engines
    - MyISAM Key Buffer
12. References
    - MySQL Performance Document
    - MySQL 5.7 was has changed to 8.0

Backlog

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. Datentypen
    - Integer - INT - Datentypen
6. Basics
    - Connection to DB + exit
    - mysql-client
    - Charset-Collations
    - server system variablen abfragen
7. Storage Engines
    - Which engine is used
8. Working with the data modelling language (DML's)
    - Working with INSERT
9. Tipps & Tricks / Do Not

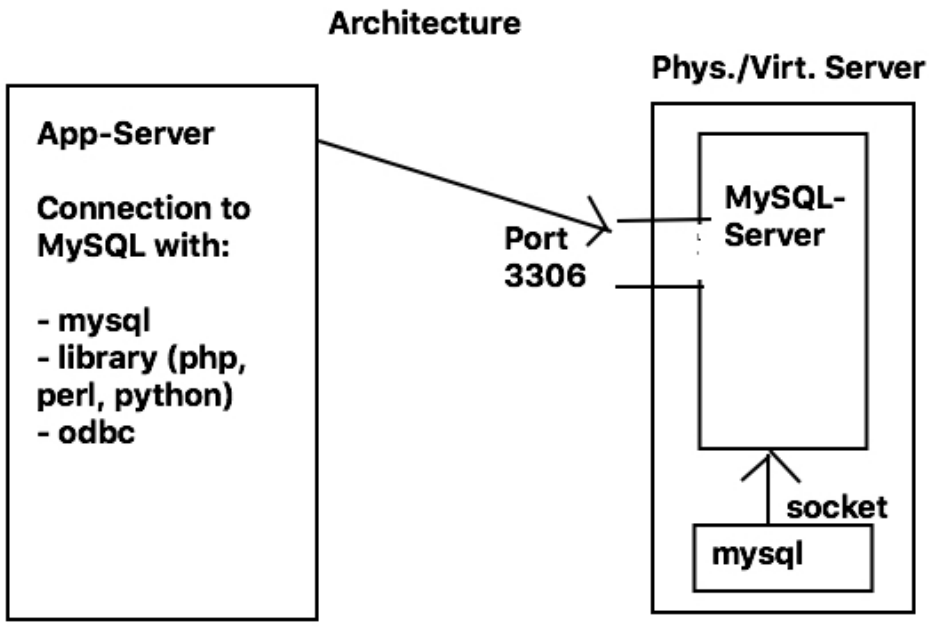
- SQL-Query im Query Tab (MySQL Workbench) direkt ausführen
- Dump/SQL-File einspielen auf der Kommandozeile - Windows
- Export Partial Columns in MYSQL with INTO OUTFILE

10. Tools
    - HeidiSQL Portable - works for windows
11. Tipps & Tricks
    - Best Practice DBAL - Kochrezept

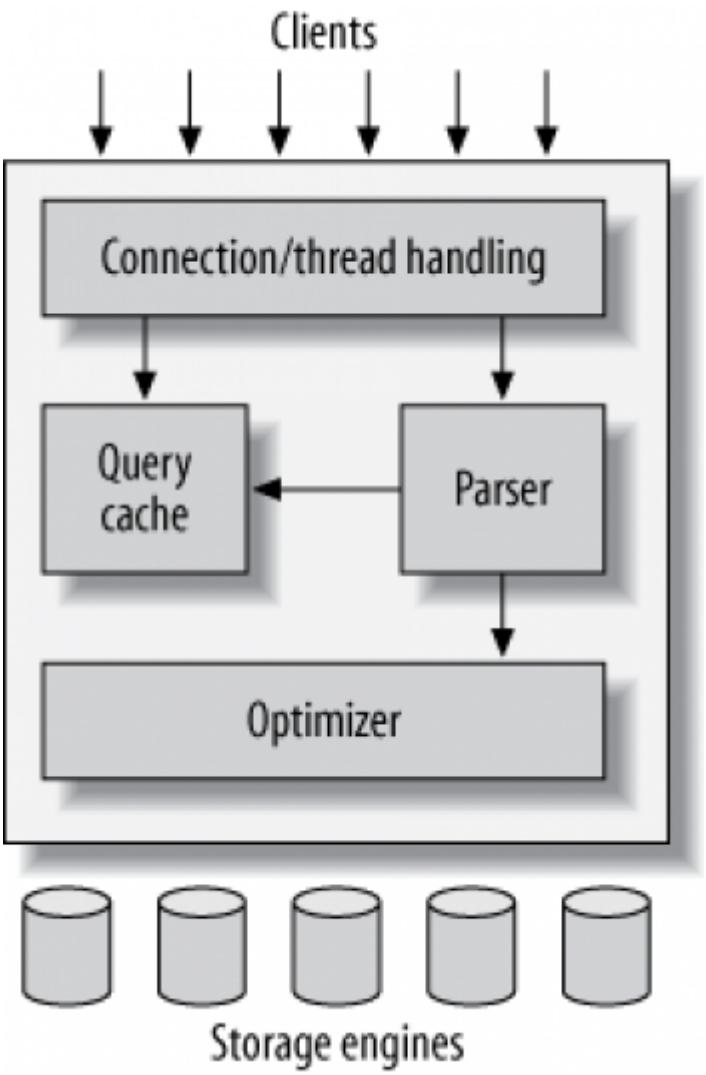
1. References
    - Examples Left Join
    - Notes on Specific MySQL Knowledge
    - Many Sakila Example Queries
    - Helpful Examples
2. Extra (Optional)
    - Schnellster Import von Daten mit csv
3. Übungen
    - Übung Update/Insert

Technical Background

Technical Structure



Process of Queries



Beispieldaten / Testserver

Sakila

```
cd /usr/src
wget https://downloads.mysql.com/docs/sakila-db.tar.gz
tar xzvf sakila-db.tar.gz

cd sakila-db
mysql < sakila-schema.sql
mysql < sakila-data.sql
```

Spendenliste

Walkthrough (Debian/Ubuntu 18.04. with mysql 5.7.)

- Complete process takes about 10 minutes

```
cd /usr/src;
apt update; apt install git;
git clone https://github.com/jmetzger/dedupe-examples.git;
cd dedupe-examples;
cd mysql_example;
## Eventually you need to enter (in mysql_example/mysql.cnf)
## Only necessary if you cannot connect to db by entering "mysql"
## password=<your_root_pw>
./setup.sh
```

Walkthrough (Debian/Ubuntu 20.04. with mysql 8)

- Complete process takes about 10 minutes

```
cd /usr/src;
apt update; apt install git;
git clone https://github.com/jmetzger/dedupe-examples.git;
cd dedupe-examples;
cd mysql_example;
## otherwice script does not work
echo "set local_infile=1" | mysql
## Eventually you need to enter (in mysql_example/mysql.cnf)
## Only necessary if you cannot connect to db by entering "mysql"
## password=<your_root_pw>
./setup.sh
```

Server Vagrant

```
## Put this in a Vagrantfile
## 1. mkdir project; cd project
## 2. Put this in a Vagrantfile
## 3. vagrant up
## 4. vagrant ssh

Vagrant.configure("2") do |config|

  config.vm.box = "ubuntu/focal64"
  config.vm.provision "shell", inline: <<-SHELL

    apt-get update
    apt-get install -y mysql-server-8.0 wget
    cd /usr/src
    wget https://downloads.mysql.com/docs/sakila-db.tar.gz
    tar xzvf sakila-db.tar.gz
    cd sakila-db
    mysql < sakila-schema.sql
    mysql < sakila-data.sql

    echo "-- Setting up external user"
    echo "CREATE USER ext@'%' identified by 'student'" | mysql
    echo "GRANT ALL PRIVILEGES ON *.* TO ext@'%'" | mysql

    echo "-- Setting mysql up for external access"
    echo "bind-address = 0.0.0.0" >> /etc/mysql/mysql.conf.d/mysqld.cnf
    systemctl restart mysql

    echo "-- Setting up contributions database - big data"
    cd /usr/src;
    apt-get install -y git
    git clone https://github.com/jmetzger/dedupe-examples.git;
    cd dedupe-examples;
    cd mysql_example;
    echo "set global local_infile = 1" | mysql
    echo "# Eventually you need to enter (in mysql_example/mysql.cnf)"
    echo '# Only necessary if you cannot connect to db by entering "mysql"'
    echo '# password=<your_root_pw>'
    ./setup.sh
    date
    echo "-- Done -  Setting up contributions database - big data"
  SHELL
end
```

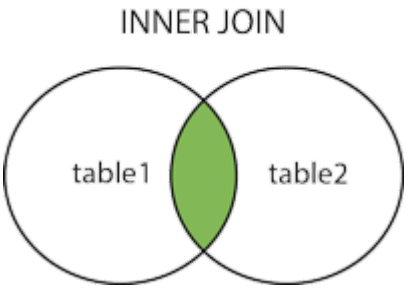
JOINS

Basics of Joins

What is a JOIN for ?

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

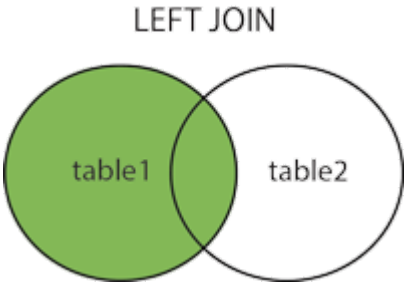
MySQL (Inner) Join



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

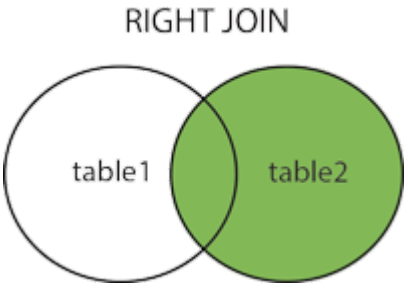
MySQL Left Join



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)

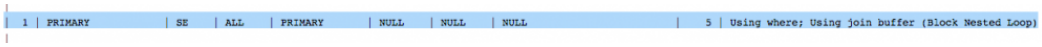
```
for each row in t1 matching range {
for each row in t2 matching reference key {
for each row in t3 {
  if row satisfies join conditions, send to client
}
}
}
```

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

CONSTRAINTS

Foreign Key Constraints - Example

```
## Walkthrough
use sakila;
create table filmcopy as select * from film;
set foreign_key_checks = 0;
truncate film;
set foreign_key_checks = 1
## WRONG
##alter table filmcopy add constraint fk_language_id references language (language_id)
alter table filmcopy add foreign key (language_id) references language (language_id);

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

UPDATE

Sophisticated Update

```
SELECT DISTINCT fc.film_id FROM filmcopy fc JOIN film_actor fa ON fc.film_id =
fa.film_id JOIN actor a ON fa.actor_id = a.actor_id where a.last_name like 'D%';

UPDATE filmcopy fc JOIN film_actor fa ON fc.film_id = fa.film_id JOIN actor a ON
fa.actor_id = a.actor_id SET rental_rate = rental_rate + 1
Query OK, 432 rows affected (0.02 sec)
Rows matched: 432  Changed: 432  Warnings: 0
```

JSON

Short Introduction working JSON

General

- Introduced since MySQL 5.7

Pitfalls

- Luke, use the index properly

Step 1: Create structure

```
create schema training;
use training;
CREATE TABLE events(
  id int auto_increment primary key,
  event_name varchar(255),
  visitor varchar(255),
  properties json,
  browser json
);
```

Step 2: Fill with data

```
INSERT INTO events(event_name, visitor,properties, browser)
VALUES (
  'pageview',
   '1',
   '{ "page": "/" }',
   '{ "name": "Safari", "os": "Mac", "resolution": { "x": 1920, "y": 1080 } }'
),
('pageview',
  '2',
  '{ "page": "/contact" }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 2560, "y": 1600 } }'
),
(
  'pageview',
  '1',
  '{ "page": "/products" }',
  '{ "name": "Safari", "os": "Mac", "resolution": { "x": 1920, "y": 1080 } }'
),
(
  'purchase',
   '3',
  '{ "amount": 200 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1600, "y": 900 } }'
),
(
  'purchase',
   '4',
```

```
  '{ "amount": 150 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1280, "y": 800 } }'
),
(
  'purchase',
  '4',
  '{ "amount": 500 }',
  '{ "name": "Chrome", "os": "Windows", "resolution": { "x": 1680, "y": 1050 } }'
);
Code language: SQL (Structured Query Language) (sql)
To pull values out of the JSON columns, you use the column path operator ( ->).
```

Get data (within select)

```
## mit hochkommas
SELECT id, browser->'$.name' browser
FROM events;

## ohne hochkommas
SELECT id, browser->'$.name' browser
FROM events;

## Die x-Auflösung
select id, browser->'$.resolution.x' browser from events
```

Get data using where

```
select id, browser->'$.name' as browser from events where browser ->"$.resolution.x" >
1600

## - no index present
explain select id, browser->'$.name' as browser from events where browser -
>"$.resolution.x" > 1600;
+----+-------------+--------+------------+------+---------------+------+---------+----
--+------+----------+-------------+
| id | select_type | table  | partitions | type | possible_keys | key  | key_len | ref
| rows | filtered | Extra       |
+----+-------------+--------+------------+------+---------------+------+---------+----
--+------+----------+-------------+
|  1 | SIMPLE      | events | NULL       | ALL  | NULL          | NULL | NULL    |
NULL |    6 |   100.00 | Using where |
+----+-------------+--------+------------+------+---------------+------+---------+----
--+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

## now creating index .
create index idx_browser on events (browser);
ERROR 3152 (42000): JSON column 'browser' supports indexing only via generated columns
on a specified JSON path.
```

Setting a specific index for json

```
alter table events add browser_resolution_x INT GENERATED ALWAYS AS (browser-
>"$.resolution.x");
-- Not working yet with index, because there is no index
explain select id, browser->'$.name' as browser from events where browser_resolution_x
= 1600;

## Set the index
create index idx_browser_resolution_x on events (browser_resolution_x)
## Now it works !!
explain select id, browser->'$.name' as browser from events where browser_resolution_x
= 1600;
```

Reference

- https://www.mysqltutorial.org/mysql-json/

Performance

* vs. specific field in field list - select

```
## You need to set up contributions database to test that
use contributions

## Variant 1: ALL Fields

mysql> select * from contributions limit 0,100000
100000 rows in set (0.25 sec)

## Variant 2: Specific field
## Try this multiple times, because the first time it is not
## in the innodb buffer (cache)

mysql> select vendor_last_name from contributions limit 0,100000;
100000 rows in set (0.04 sec)

## Result: Variant 2 wins over Variant 1
## The difference between these 2 is factor 5x in my case
```

Möglichst keine Funktion in where (spalte) verwenden

```
## Bad for performance

## Bad


## Good
```

SQL-Rewrite Pager- Subselect

```
explain SELECT film_id, description FROM sakila.film ORDER BY title LIMIT 50, 5;
+----+-------------+-------+------------+------+---------------+------+---------+-----
-+------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref
| rows | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+-----
-+------+----------+----------------+
|  1 | SIMPLE      | film  | NULL       | ALL  | NULL          | NULL | NULL    | NULL
| 1000 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+-----
-+------+----------+----------------+
1 row in set, 1 warning (0.00 sec)

## Achtung auf title ist ein index sonst geht es nicht
SELECT film.film_id, film.description
FROM sakila.film
INNER JOIN (
    SELECT film_id FROM sakila.film
    ORDER BY title LIMIT 50, 5
) AS lim USING(film_id);

SELECT film.film_id, film.description FROM sakila.film INNER JOIN (     SELECT film_id
FROM sakila.film     ORDER BY title
LIMIT 50, 5 ) AS lim USING(film_id);

explain SELECT film.film_id, film.description FROM sakila.film INNER JOIN (     SELECT
film_id FROM sakila.film     ORDER B
Y title LIMIT 50, 5 ) AS lim USING(film_id);

explain SELECT film.film_id, film.description FROM sakila.film INNER JOIN (     SELECT
film_id FROM sakila.film     ORDER BY title LIMIT 50, 5 ) AS lim USING(film_id);
+----+-------------+------------+------------+--------+---------------+-----------+---
------+-------------+------+----------+-------------+
| id | select_type | table      | partitions | type   | possible_keys | key       |
key_len | ref         | rows | filtered | Extra       |
+----+-------------+------------+------------+--------+---------------+-----------+---
------+-------------+------+----------+-------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL    | NULL          | NULL      |
NULL    | NULL        |   55 |   100.00 | NULL        |
|  1 | PRIMARY     | film       | NULL       | eq_ref | PRIMARY       | PRIMARY   | 2
| lim.film_id |    1 |   100.00 | NULL        |
|  2 | DERIVED     | film       | NULL       | index  | NULL          | idx_title |
514     | NULL        |   55 |   100.00 | Using index |
+----+-------------+------------+------------+--------+---------------+-----------+---
------+-------------+------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)
```

Analyzing Slow Queries / Indexes

Find indexes

Show index from table

```
create database showindex;
use showindex;
CREATE TABLE `people` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `first_name` varchar(25) DEFAULT NULL,
  `last_name` varchar(25) DEFAULT NULL,
  `passcode` mediumint(8) unsigned DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `idx_passcode` (`passcode`),
  KEY `idx_first_name_last_name` (`first_name`,`last_name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
show index from people
```

Show create table

```
show create table peple
```

show index from

```
show index from contributions
```

Create Index/Delete/Drop Index

```
create index idx_vendor_state on contributions (vendor_state);
```

Drop/Delete Index

```
drop index idx_last_name_first_name on customer;
```

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%';
```

Explain

Einfacher Fall

```
explain select * from actor
```

Erweiterter Fall

```
explain extended select * from user
show warnings
```

Anzeigen der Partitions

```
explain partitions select * from actor
```

Ausgabe im JSON-Format

```
## Hier gibt es noch zusätzliche Informationen
explain format=json select * from actor
```

```
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

profiling-get-time-for-execution-of.query

- Get better values, how long queries take

Example

```
set profiling = 1
-- Step 2 - Execute query
select last_name as gross from donors where last_name like lower('WILLI%')

## Step 3 - Show profiles
show profiles;
+----------+------------+-------------------------------------------------------------------
----------------------+
| Query_ID | Duration   | Query
|
+----------+------------+-------------------------------------------------------------------
----------------------+
|        1 | 0.01993525 | select last_name as gross from donors where last_name like
lower('WILLI%')        |
4 rows in set, 1 warning (0.00 sec)

## Step 4 - Show profile for a specific query
mysql> show profile for query 1;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000062 |
| checking permissions | 0.000006 |
| Opening tables       | 0.000021 |
| init                 | 0.000017 |
| System lock          | 0.000007 |
| optimizing           | 0.000007 |
| statistics           | 0.000083 |
| preparing            | 0.000012 |
| executing            | 0.000004 |
| Sending data         | 0.022251 |
| end                  | 0.000005 |
| query end            | 0.000008 |
| closing tables       | 0.000007 |
| freeing items        | 0.001792 |
| cleaning up          | 0.000016 |
+----------------------+----------+
15 rows in set, 1 warning (0.00 sec)
```

Kein function in where verwenden

1. No function in where (column_name)

```
## Never use a function for the column name in where
## e.g.
select * from donors where upper(last_name) like 'Willia%'
```

Why ?

- Not index can be used

```
## Not filtering possible by indx -> possible_keys -> NULL
 explain select last_name from donors where upper(last_name) like 'WILLI%';
+----+-------------+--------+------------+-------+---------------+-------------------
+---------+------+--------+----------+--------------------------+
| id | select_type | table  | partitions | type  | possible_keys | key               |
key_len | ref  | rows   | filtered | Extra                    |
+----+-------------+--------+------------+-------+---------------+-------------------
+---------+------+--------+----------+--------------------------+
|  1 | SIMPLE      | donors | NULL       | index | NULL          | donors_donor_info |
687     | NULL | 701948 |   100.00 | Using where; Using index |
+----+-------------+--------+------------+-------+---------------+-------------------
+---------+------+--------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

Optimizer-hints (and why you should not use them)

Tell the optimizer what to do and what not to do

- https://dev.mysql.com/doc/refman/5.7/en/optimizer-hints.html#optimizer-hints-syntax

This one is good for debugging / do not use index at all

```
explain select vendor_city from contributions use index() where vendor_city like 'S%'
```

Query-Plans aka Explains

- Query Plans are the same as Query Execution Plans (QEP's)
- You will see the Query Plan's with explain

Example

```
mysql> explain select * from recipients where recipient_id > 1 and recipient_id < 5;
+----+-------------+------------+------------+-------+---------------+---------+------
---+------+------+----------+-------------+
| id | select_type | table      | partitions | type  | possible_keys | key     |
key_len | ref  | rows | filtered | Extra       |
+----+-------------+------------+------------+-------+---------------+---------+------
---+------+------+----------+-------------+
|  1 | SIMPLE      | recipients | NULL       | range | PRIMARY       | PRIMARY | 4
| NULL |    1 |   100.00 | Using where |
+----+-------------+------------+------------+-------+---------------+---------+------
---+------+------+----------+-------------+
1 row in set, 1 warning (0.01 sec)
```

Output-Format json

```
-- includes costs
EXPLAIN format=json SELECT * from audit_log WHERE yr in (2011,2012);
```

Select_Type

- simple = one table

Types (in order of performance

system

```
Only one row in table is present (only one insert)
```

Query Pläne und die Key-Länge

Example

```
 select table_schema,table_name,character_set_name,column_name from
information_schema.columns where table_name = 'donors'
and column_name = 'city';
+---------------+------------+--------------------+-------------+
| table_schema  | table_name | character_set_name | column_name |
+---------------+------------+--------------------+-------------+
| contributions | donors     | utf8               | city        |
+---------------+------------+--------------------+-------------+
1 row in set (0.00 sec)
```

```
mysql> describe donors;
+--------------------+-------------+------+-----+---------+----------------+
| Field              | Type        | Null | Key | Default | Extra          |
+--------------------+-------------+------+-----+---------+----------------+
| donor_id           | int(11)     | NO   | PRI | NULL    | auto_increment |
| last_name          | varchar(70) | YES  | MUL | NULL    |                |
| first_name         | varchar(35) | YES  |     | NULL    |                |
| address_1          | varchar(35) | YES  |     | NULL    |                |
| address_2          | varchar(36) | YES  |     | NULL    |                |
| city               | varchar(20) | YES  | MUL | NULL    |                |
| state              | varchar(15) | YES  |     | NULL    |                |
| zip                | varchar(11) | YES  |     | NULL    |                |
| employer           | varchar(70) | YES  |     | NULL    |                |
| occupation         | varchar(40) | YES  |     | NULL    |                |
| last_name_reversed | varchar(70) | YES  | MUL | NULL    |                |
+--------------------+-------------+------+-----+---------+----------------+
11 rows in set (0.00 sec)
```

```
## only the first part of the combined index is used, because 213 bytes is a
varchar(70) for utf8 - characters in index
## utf8 takes up to 3 bytes per character.
mysql> explain select first_name,last_name from donors where last_name like 'Wi%';
+----+-------------+--------+------------+-------+-------------------+----------------
---+---------+------+-------+----------+-------------------------+
| id | select_type | table  | partitions | type  | possible_keys     | key
| key_len | ref  | rows  | filtered | Extra                   |
+----+-------------+--------+------------+-------+-------------------+----------------
---+---------+------+-------+----------+-------------------------+
|  1 | SIMPLE      | donors | NULL       | range | donors_donor_info |
donors_donor_info | 213     | NULL | 18722 |   100.00 | Using where; Using index |
+----+-------------+--------+------------+-------+-------------------+----------------
---+---------+------+-------+----------+-------------------------+
1 row in set, 1 warning (0.00 sec)

## both last_name and first_name are used to filter
## 3 bytes + 70x3 (varchar(70)) + 3Bytes + 35x3 (varchar(35)) = 321
mysql> explain select first_name,last_name from donors where last_name like 'Williams'
and first_name like 'A%';
```

```
+----+-------------+--------+------------+-------+-------------------+----------------
---+---------+------+------+----------+--------------------------+
| id | select_type | table  | partitions | type  | possible_keys     | key
| key_len | ref  | rows | filtered | Extra                     |
+----+-------------+--------+------------+-------+-------------------+----------------
---+---------+------+------+----------+--------------------------+
|  1 | SIMPLE      | donors | NULL       | range | donors_donor_info |
donors_donor_info | 321     | NULL |   46 |   100.00 | Using where; Using index |
+----+-------------+--------+------------+-------+-------------------+----------------
---+---------+------+------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql>
```

Index und Likes

1. like 'Will%' - Index works

explain select last_name from donors where last_name like 'Will%';

2. like '%iams' - Index does not work

```
-- because like starts with a wildcard
explain select last_name from donors where last_name like '%iams';
```

3. How to fix 3, if you are using this often ?

```
## Walkthrough
## Step 1: modify table
alter table donors add last_name_reversed varchar(70) GENERATED ALWAYS AS
(reverse(last_name));
create index idx_last_name_reversed on donors (last_name_reversed);

## besser - Variante 2 - untested
alter table donors add last_name_reversed varchar(70) GENERATED ALWAYS AS
(reverse(last_name)), add index idx_last_name_reversed on donors (last_name_reversed);

## Step 2: update table - this take a while
update donors set last_name_reversed = reversed(last_name)
## Step 3: work with it
select last_name,last_name_reversed from donor where last_name_reversed like
reverse('%iams');
```

```
## Version 2 with pt-online-schema-change
```

Index und Joins

Take a look which order the optimizer uses

With date

```
-- Using a date which has no index
-- Needs to do a table scan
explain select c.* from contributions c join donors d using (donor_id) join recipients
r using (recipient_id) where c.date_recieved > '1999-12-01' and c.date_recieved <
'2000-07-01';
+----+-------------+-------+------------+--------+-------------------------+---------
+---------+-------------------------------+---------+----------+-------------+
| id | select_type | table | partitions | type   | possible_keys           | key     |
key_len | ref                           | rows    | filtered | Extra       |
+----+-------------+-------+------------+--------+-------------------------+---------
+---------+-------------------------------+---------+----------+-------------+
|  1 | SIMPLE      | c     | NULL       | ALL    | donor_idx,recipient_idx | NULL    |
NULL    | NULL                          | 2028240 |    11.11 | Using where |
|  1 | SIMPLE      | r     | NULL       | eq_ref | PRIMARY                 | PRIMARY |
4       | contributions.c.recipient_id |       1 |   100.00 | Using index |
|  1 | SIMPLE      | d     | NULL       | eq_ref | PRIMARY                 | PRIMARY |
4       | contributions.c.donor_id     |       1 |   100.00 | Using index |
+----+-------------+-------+------------+--------+-------------------------+---------
+---------+-------------------------------+---------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)

60626 rows in set (7.22 sec)
```

With date and filter on donor

```
 explain select c.*,d.last_name from contributions c join donors d using (donor_id)
join recipients r using (recipient_id)
where c.date_recieved > '1999-12-01' and c.date_recieved < '2000-07-01' and
d.last_name like 'A%';
+----+-------------+-------+------------+--------+-------------------------+--------
-----------+---------+-------------------------------+-------+----------+-------------
------------+
| id | select_type | table | partitions | type   | possible_keys           | key
| key_len | ref                           | rows  | filtered | Extra
|
+----+-------------+-------+------------+--------+-------------------------+--------
-----------+---------+-------------------------------+-------+----------+-------------
------------+
|  1 | SIMPLE      | d     | NULL       | range  | PRIMARY,donors_donor_info |
donors_donor_info | 213     | NULL                          | 65894 |   100.00 | Using
where; Using index |
|  1 | SIMPLE      | c     | NULL       | ref    | donor_idx,recipient_idx   |
donor_idx         | 5       | contributions.d.donor_id     |     2 |    11.11 | Using
where              |
|  1 | SIMPLE      | r     | NULL       | eq_ref | PRIMARY                 | PRIMARY
| 4       | contributions.c.recipient_id |     1 |   100.00 | Using index
```

```
|
+----+-------------+-------+------------+--------+---------------------------+--------
-----------+---------+-------------------------------+-------+----------+-------------
------------+
3 rows in set, 1 warning (0.00 sec)
```

With date and filter on donor, less specific

```
select c.*,d.* from contributions c join donors d using (donor_id) join recipients r
using (recipient_id) where c.date_recieved > '1999-12-01' and c.date_recieved < '2000-
07-01' and d.last_name like 'A%';
 explain select c.*,d.* from contributions c join donors d using (donor_id) join
recipients r using (recipient_id) where c.date_recieved > '1999-12-01' and
c.date_recieved < '2000-07-01' and d.last_name like 'A%';
+----+-------------+-------+------------+--------+---------------------------+--------
-----------+---------+-------------------------------+-------+----------+-------------
---------+
| id | select_type | table | partitions | type   | possible_keys             | key
| key_len | ref                          | rows  | filtered | Extra                 |
+----+-------------+-------+------------+--------+---------------------------+--------
-----------+---------+-------------------------------+-------+----------+-------------
---------+
|  1 | SIMPLE      | d     | NULL       | range  | PRIMARY,donors_donor_info |
donors_donor_info | 213     | NULL                          | 65894 |   100.00 | Using
index condition |
|  1 | SIMPLE      | c     | NULL       | ref    | donor_idx,recipient_idx   |
donor_idx         | 5       | contributions.d.donor_id     |     2 |    11.11 | Using
where           |
|  1 | SIMPLE      | r     | NULL       | eq_ref | PRIMARY                   | PRIMARY
| 4       | contributions.c.recipient_id |     1 |   100.00 | Using index           |
+----+-------------+-------+------------+--------+---------------------------+--------
-----------+---------+-------------------------------+-------+----------+-------------
---------+
3 rows in set, 1 warning (0.00 sec)
```

With date and filter on donor and filter on recipient

```
mysql> explain select c.*,d.last_name,r.* from contributions c join donors d using
(donor_id) join recipients r using (recipient_
id) where c.date_recieved > '1999-12-01' and c.date_recieved < '2000-07-01' and
d.last_name like 'A%' and r.name like 'Cit%';
+----+-------------+-------+------------+--------+---------------------------+--------
-------+---------+-------------------------------+------+----------+-------------+
| id | select_type | table | partitions | type   | possible_keys             | key
| key_len | ref                          | rows | filtered | Extra       |
+----+-------------+-------+------------+--------+---------------------------+--------
-------+---------+-------------------------------+------+----------+-------------+
|  1 | SIMPLE      | r     | NULL       | ALL    | PRIMARY                   | NULL
| NULL    | NULL                          | 6063 |    11.11 | Using where |
|  1 | SIMPLE      | c     | NULL       | ref    | donor_idx,recipient_idx   |
recipient_idx | 5       | contributions.r.recipient_id |  305 |    11.11 | Using where
```

```
|
|  1 | SIMPLE      | d     | NULL       | eq_ref | PRIMARY,donors_donor_info | PRIMARY
| 4       | contributions.c.donor_id     |    1 |     9.39 | Using where |
+----+-------------+-------+------------+--------+---------------------------+--------
-------+---------+-------------------------------+------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)
```

Find out cardinality without index

Find out cardinality without creating index

```
select count(distinct donor_id) from contributions;
```

```
select count(distinct(vendor_city)) from contributions;
+------------------------------+
| count(distinct(vendor_city)) |
+------------------------------+
|                         1772 |
+------------------------------+
1 row in set (4.97 sec)
```

Index and Functions

No index can be used on an index:

```
explain select * from actor where upper(last_name) like 'A%';
+----+-------------+-------+------------+------+---------------+------+---------+-----
-+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref
| rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+-----
-+------+----------+-------------+
|  1 | SIMPLE      | actor | NULL       | ALL  | NULL          | NULL | NULL    | NULL
|  200 |   100.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+-----
-+------+----------+-------------+
```

Workaround with virtual columns (possible since mysql 5.7)

```
## 1. Create Virtual Column with upper
alter table sakila add idx_last_name_upper varchar(45) GENERATED ALWAYS AS
upper(last_name);
## 2. Create an index on that column
create index idx_last_name_upper on actor (last_name_upper);
```

Workaround with persistent/virtual columns (MariaDB)

```
mysql> alter table actor add column last_name_upper varchar(45) as (upper(last_name))
PERSISTENT ;
mysql> insert into actor (first_name,last_name,last_name_upper) values
('Max','Mustermann','MUSTERMANN');
mysql> select * from actor order by actor_id desc limit 1;
mysql> -- setting index
mysql> create index idx_last_name_upper on actor (last_name_upper);
Query OK, 0 rows affected (0.007 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> -- to use index we need to avoid the function in where
mysql> explain select * from actor where last_name_upper like 'WI%' \G
```

Tools

Percona Toolkit

Walkthrough (Ubuntu 20.04)

```
## Howto
## https://www.percona.com/doc/percona-toolkit/LATEST/installation.html

## Step 1: repo installieren mit deb -paket
wget https://repo.percona.com/apt/percona-release_latest.focal_all.deb
apt update
apt install -y curl
dpkg -i percona-release_latest.focal_all.deb
apt update
apt install -y percona-toolkit
```

Walkthrough (Debian 10)

```
sudo apt update
sudo apt install -y wget gnupg2 lsb-release curl
cd /usr/src
wget https://repo.percona.com/apt/percona-release_latest.generic_all.deb
dpkg -i percona-release_latest.generic_all.deb
apt update
apt install -y percona-toolkit
```

```
sudo apt update; sudo apt install -y wget gnupg2 lsb-release curl; cd /usr/src; wget
https://repo.percona.com/apt/percona-release_latest.generic_all.deb; dpkg -i percona-
release_latest.generic_all.deb; apt update; apt install -y percona-toolkit
```

pt-query-digest - analyze slow logs

Requires

- Install percona-toolkit

Usage

```
## first enable slow_query_log
set global slow_query_log = on
set global long_query_time = 0.2
## to avoid, that i have to reconnect with new session
set session long_query_time = 0.2

## produce slow query - for testing
select * from contributions where vendor_last_name like 'W%';
mysql > quit

##
cd /var/lib/mysql
## look for awhile wih -slow.log - suffix
pt-query-digest mysql-slow.log > /usr/src/report-slow.txt
less report-slow.txt
```

pt-online-schema-change howto

Requirements

- Install percona-toolkit

Documentation

- https://www.percona.com/doc/percona-toolkit/3.0/pt-online-schema-change.html

What does it do ?

```
## Altering table without blocking them
## Do a dry-run beforehand
pt-online-schema-change --alter "ADD INDEX idx_city (city)" --dry-run
D=contributions,t=donors
##
pt-online-schema-change --alter "ADD INDEX idx_city (city)" --execute
D=contributions,t=donors
```

With foreign - keys

```
 # first try
 pt-online-schema-change --alter "add column remark varchar(150)" D=sakila,t=actor --
alter-foreign-keys-method=auto --dry-run
 # then run
 pt-online-schema-change --alter "add column remark varchar(150)" D=sakila,t=actor --
alter-foreign-keys-method=auto --execute
```

Example sys-schema and Reference

```
mysql> select * from sys.host_summary\G
*************************** 1. row ***************************
                  host: localhost
            statements: 1347
     statement_latency: 7.55 m
 statement_avg_latency: 336.50 ms
           table_scans: 15
              file_ios: 612857
       file_io_latency: 1.66 m
   current_connections: 1
     total_connections: 7
          unique_users: 1
        current_memory: 0 bytes
total_memory_allocated: 0 bytes
1 row in set (0.01 sec)
```

Ref:

- https://github.com/mysql/mysql-sys/blob/master/README.md

Storage Engines

MyISAM Key Buffer

- http://www.mysqlab.net/knowledge/kb/detail/topic/myisam/id/7200

References

MySQL Performance Document

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

MySQL 5.7 was has changed to 8.0

- https://severalnines.com/database-blog/moving-mysql-57-mysql-80-what-you-should-know

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;
```

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
```

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

Tipps & Tricks

Best Practice DBAL - Kochrezept

Ausgangsituation

```
DBAL erstellt query
Query ist langsam
Was tun ?
```

Steps 1+2

```
### An das SQL-Statement rankommen.


### 1. Analyse, warum ist Query langsam (kann ich von aussen etwas tun, um die Query
zu beeinflussen ohne sie zu ändern
explain <sql-statement>

### Sichtprüfung, fehlen Keys ??
##
##explain select vendor_city,vendor_state from contributions where vendor_first_name
like 'D%';
##+----+-------------+---------------+------------+------+---------------+------+-----
----+------+---------+----------+-------------+
##| id | select_type | table         | partitions | type | possible_keys | key  |
key_len | ref  | rows    | filtered | Extra       |
##+----+-------------+---------------+------------+------+---------------+------+-----
----+------+---------+----------+-------------+
##|  1 | SIMPLE      | contributions | NULL       | ALL  | NULL          | NULL | NULL
| NULL | 2028402 |    11.11 | Using where |
##+----+-------------+---------------+------------+------+---------------+------+-----
----+------+---------+----------+-------------+
## Gibt es bei dem type einen Eintrag ALL.

### Oder da ist ein key drin, aber der wird garnicht verändert.

### Index setzen
```

Step 3.

```
Query analysieren -> Schritt debuggen.
1) Links -> Rechts: Entweder von Links (sprich: was sind die ersten Daten, die ich
brauche, welche dann, welche dann usw.)
```

```
2) Rechts -> Links -> immer mehr wegnehmen und gucken, ob es schneller wird.
```

Step 4:

```
index hint.
```

Step 5: Customized Query

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

Helpful Examples

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

Extra (Optional)

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');
```