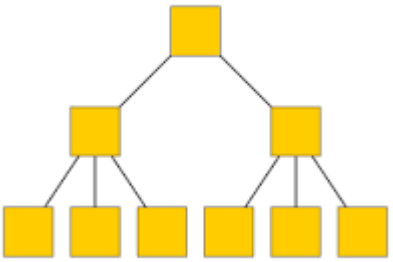
Dokumentation ÜK 105 - Noah

DATA TYPS. FUNCTIONS.

Datenbank Typen:

Hierarchische Datenbank:

- Die hierarchische Datenbank organisiert Daten in einer Baumstruktur, bei der jede Entität eine übergeordnete und mehrere untergeordnete Entitäten haben kann.
- Die Daten werden in Form von Records und Feldern gespeichert.
- Beziehungen zwischen Entitäten werden durch Verknüpfungen auf verschiedenen Ebenen hergestellt, wobei eine übergeordnete Entität mit mehreren untergeordneten Entitäten verbunden sein kann.
- Der Zugriff auf Daten erfolgt hauptsächlich über einen Top-Down-Ansatz, wobei die Navigation von der Wurzel des Baums zu den Blättern erfolgt.
- Hierarchische Datenbanken wurden hauptsächlich in den frühen Tagen der Datenbankentwicklung verwendet und finden heute nur noch begrenzte Anwendung.



Relationale und Objektrelationale Datenbanken:

- Die Daten werden in Tabellen organisiert, wobei jede Tabelle aus Zeilen und Spalten besteht. Jede Zeile repräsentiert einen Datensatz, und jede Spalte enthält einen bestimmten Datentyp.

- Beziehungen zwischen Tabellen werden durch Schlüsselbeziehungen hergestellt, bei denen Primärschlüssel und Fremdschlüssel verwendet werden.

- Abfragen werden mit der strukturierten Abfragesprache SQL durchgeführt.

- Objektrelationale Datenbanken erweitern das relationale Modell um objektorientierte Konzepte. Sie ermöglichen die Speicherung und Verwaltung komplexer Datenstrukturen
- **Datenorganisation**: Relationale Datenbanken organisieren Daten in Tabellen, die aus Zeilen und Spalten bestehen. Dies ermöglicht eine strukturierte Speicherung und 21 Verwaltung von Daten.

- **Integritätssicherung**: Relationale Datenbanken bieten Mechanismen zur Sicherstellung der Datenintegrität, z. B. durch Primärschlüssel, Fremdschlüssel und Constraints. Dadurch werden Datenkonsistenz und -genauigkeit gewährleistet.

- **Beziehungen**: Relationale Datenbanken ermöglichen die Definition und Verwaltung von Beziehungen zwischen Tabellen. Dadurch können Daten auf logische und effiziente Weise verknüpft werden.

- **Datenabfragen**: Relationale Datenbanken unterstützen leistungsstarke Abfragemöglichkeiten, die es ermöglichen, komplexe Datenabfragen durchzuführen. Die strukturierte Abfragesprache SQL wird verwendet, um Datenabfragen zu formulieren und auszuführen.

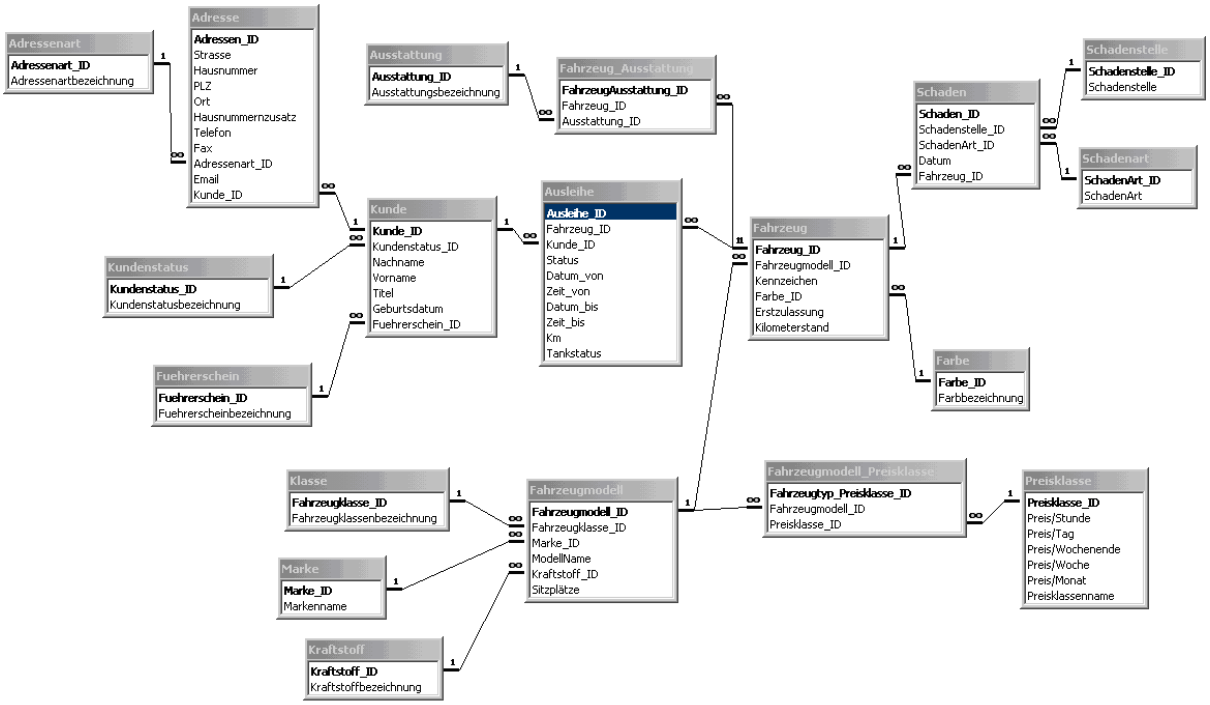
- **Skalierbarkeit**: Relationale Datenbanken sind in der Lage, mit wachsenden Datenmengen und Benutzeranforderungen zu skalieren. Durch Indexierung, Optimierungstechniken und Hardware-Upgrades können sie grosse Datenmengen effizient verarbeiten.

- **Transaktionsverwaltung**: Relationale Datenbanken unterstützen das Konzept von Transaktionen, die sicherstellen, dass Datenbankoperationen entweder vollständig ausgeführt werden oder gar nicht. Dies ermöglicht die Wahrung der Datenkonsistenz und -zuverlässigkeit.

- **Daten Zugriffskontrolle:** Relationale Datenbanken bieten Mechanismen zur Zugriffskontrolle, um sicherzustellen, dass nur autorisierte Benutzer auf Daten zugreifen können. Berechtigungen können auf der Tabellenspalten- und Zeilenebene festgelegt werden.

- **Datenunabhängigkeit**: Relationale Datenbanken ermöglichen eine Trennung von Daten und Anwendungslogik. Änderungen an der Datenstruktur haben keinen direkten Einfluss auf die Anwendungsprogramme, was die Wartung und Weiterentwicklung erleichtert.
-
- wie Objekte, Vererbung und Polymorphismus.

- Objektrelationale Datenbanken bieten eine höhere Flexibilität bei der Modellierung von Daten, insbesondere für komplexe Anwendungen.

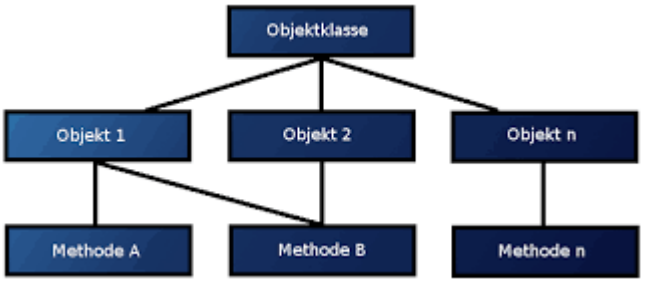


können.

- Objektorientierte Datenbanken unterstützen Konzepte wie Vererbung, Polymorphismus und Datenkapselung.

- Abfragen werden in der Regel mit objektorientierten Abfragesprachen oder objektrelationalem Mapping durchgeführt.

- Objektorientierte Datenbanken eignen sich gut für Anwendungen, bei denen komplexe Datenstrukturen und komplexe Beziehungen zwischen den Daten vorhanden sind, wie beispielsweise in CAD-Systemen oder objektorientierten Anwendungen.



Wichtige SQL Befehle:

Alle user anzeigen `SELECT User FROM mysql.user;`

LOGIN: mysql.exe -u root -p localhost

SHOW DATABASES; USE mysql; SHOW TABLES; DECRIBE oder DESC table;

script ausführen: SOURCE \User\noah\documents\script1.sql;

AUTO_INCREMENT: ist eine Eigenschaft oder Funktion, die in den meisten relationalen Datenbankmanagementsystemen (DBMS) verfügbar ist. Sie ermöglicht es, automatisch eindeutige und inkrementierende Werte für eine Spalte zu generieren, in der eine Primärschlüssel- oder eindeutige Werte benötigt werden.

Normalisiernung:

1. Normalform (1NF)

ist dann gegeben, wenn alle Attributwerte atomar sind. Dabei ist ein Attribut dann atomar, wenn es weder aus leineren Einheiten zusammengesetzt ist, noch mehrwertig ist.

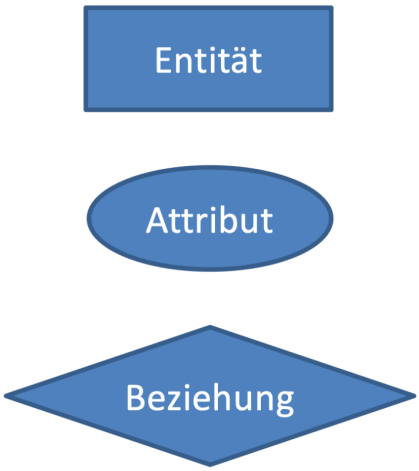
2. Normalform (2NF)

Ist dann gegeben, wenn sie in der ersten Normalform ist und jedes Nichtschlüsselattribut von jedem Schlüssel voll funktional abhängig ist. Konkret heisst das, dass Nichtschlüsselattribute vom kompletten Schlüssel und nicht nur von einem Teil des Schlüssels abhängen müssen.

3. Normalform (3NF)

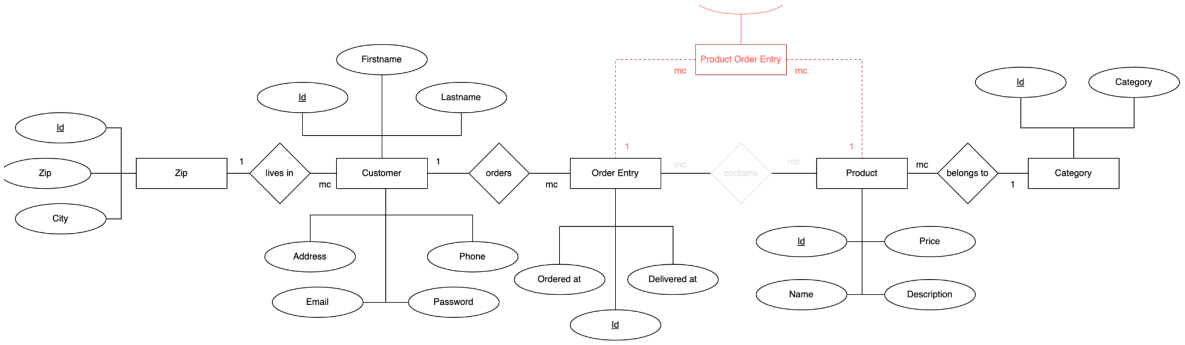
Ist dann gegeben, wenn sie in der ersten und zweiten Normalform vorliegt und kein Nichtschlüsselattribut von (irgend)einem Schlüssel transitiv abhängt. Um eine transitive Abhängigkeit also auszuschließen, dürfen alle Nichtschlüsselattribute nicht von anderen Nichtschlüsselattributen abhängig sein.

ERM:



Kardinalitäten

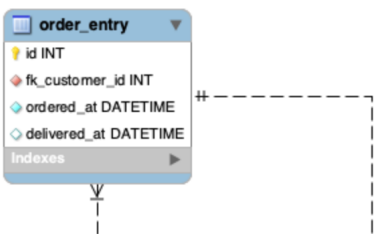
- `1` – Genau 1
- `c` – 0 oder 1
- `m` – 1 oder mehrere
- `mc` – 0, 1 oder mehrere



DBM (Datenbank Modell):

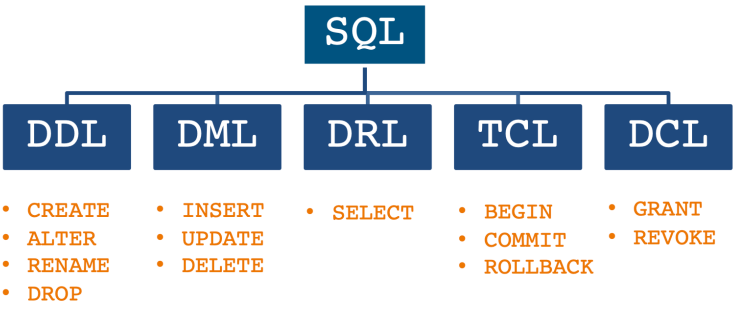
Konventionen

- Tabellen- und Feldernamen werden geschrieben
- auf Englisch
- Klein
- Singular
- Identifikationsschlüssel allgemein "id"
- Fremdschlüssel mit dem Prefix "fk_"
- Wortverbindungen mit _



SQL – Sprachelemente:

- DDL (DATA DEFINITION)
- DML (DATA MANIPULATION)
- DRL (DATA RETRIEVAL)
- TCL (TRANSACTION CONTROL)
- DCL (DATA CONTROL)



DDL:

Tabelle erstellen:

```
CREATE TABLE <table_name> (
        <attribut_name> <datatype> <CONSTRAINTS>,
        <attribut_name> <datatype> <CONSTRAINTS>,
        ...
        <CONSTRAINTS, KEYS, INDEXES>
);
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <attribut_name>
  - Name des Attributs (gemäss Konventionen)
- <datatype>
  - Datentyp (gemäss Referenz)
- <CONSTRAINTS>
  - (NOT) NULL, UNIQUE, DEFUALT
- <CONSTRAINTS, KEYS, INDEXES>
  - PRIMARY KEY(…)
  - FOREIGN KEY(…) REFERENCES <table_name>(..)

Attribut hinzufügen:

```
ALTER TABLE <table_name> ADD <attribut_name> <datatype> <CONSTRAINTS, KEYS, INDEXES>;

mysql> ALTER TABLE person_course_execution ADD confirmed BOOLEAN DEFAULT 0;
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <attribut_name>
  - Name des Attributs (gemäss Konventionen)
- <datatype>
  - Datentyp (gemäss Referenz)
- <CONSTRAINTS>
  - (NOT) NULL, UNIQUE, DEFUALT
- <CONSTRAINTS, KEYS, INDEXES>
  - PRIMARY KEY(…)
  - FOREIGN KEY(…) REFERENCES ON <table_name>(..)

Attribut ändern:

```
ALTER TABLE <table_name> MODIFY <attribut_name> <NEW_datatype> <NEW_CONSTRAINTS,
NEW_KEYS, NEW_INDEXES>;

mysql> ALTER TABLE person_course_execution MODIFY confirmed TINYINT DEFAULT 0;
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <attribut_name>
  - Name des Attributs (gemäss Konventionen)
- <NEW_datatype>
  - Datentyp (gemäss Referenz)
- <NEW_CONSTRAINTS>
  - (NOT) NULL, UNIQUE, DEFUALT
- <NEW_CONSTRAINTS, NEW_KEYS, NEW_INDEXES>
  - PRIMARY KEY(…)
  - FOREIGN KEY(…) REFERENCES ON <table_name>(..)

Attribut umbenennen:

```
ALTER TABLE <table_name> CHANGE <attribut_name> <NEW_attribut_name> <datatype>;

mysql> ALTER TABLE person_course_execution CHANGE confirmed isConfirmed TINYINT;
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <attribut_name>
  - Name des Attributs (gemäss Konventionen)
- <NEW_attribute_name>
  - Neuer Name des Attributs (gemäss Konventionen)
- <datatype>
  - Datentyp des Attributs

Attribut löschen:

```
ALTER TABLE <table_name> DROP <attribut_name>;

mysql> ALTER TABLE person_course_execution DROP confirmed;
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <attribut_name>
  - Name des Attributs (gemäss Konventionen)

Foreign Key löschen:

```
mysql> SHOW CREATE TABLE person_course_execution;

ALTER TABLE <table_name> DROP FOREIGN KEY <constraint_name>;

mysql> ALTER TABLE person_course_execution DROP FOREIGN KEY
person_course_execution_ibfk_1;
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <constraint_name>
  - Name des Constraint (gemäss SHOW CREATE TABLE)

Foreign Key hinzufügen:

```
ALTER TABLE <table_name> ADD [CONSTRAINT <constraint_name>] FOREIGN
        KEY(<attribut_name_of_foreign_key>) REFERENCES
        <reference_target_table>(<reference_target_attribut>);

mysql> ALTER TABLE person_course_execution ADD [CONSTRAINT
person_course_execution_ibfk_1] FOREIGN KEY(fk_participant_id) REFERENCES person(id);
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <constraint_name>
  - Name des Constraint (<table_name>_ibfk_1)
- <attribut_name_of_foreign_key>
  - Name des Attributs, der als Fremdschlüssel definiert werden sollte
- <reference_target_table>
  - Name der Zieltabelle
- <reference_target_attribut>
  - Name des Zielattributs auf der Zieltabelle

DML:

Datensätze hinzufügen:

```
INSERT INTO <table_name> (<attribut_name>, <attribut_name>, …) VALUES
        (<value1>, <value2>, <value3>),
        (<value1>, <value2>, <value3>),
        …;

mysql> INSERT INTO course (name, description, price) VALUES
        ('101', 'Webseite erstellen und veröffentlichen', 300.00),
        ('106', 'Datenbanken mit SQL bearbeiten', 500.00),
        ('114', 'Codierungs-Kompression-Verschlüsselungsverfahr', 700.00);
```

Datensätze anpassen:

```
UPDATE <table_name> SET
        <attribut_name1> = <value1>
        [, <attribut_name2> = <value2>]
WHERE <attribut_name> = <value>;
```

```
mysql> UPDATE person SET firstname = 'Markus' WHERE firstname = 'Vorname_S1';
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <attribute_name>
  - Name des Attributs
- <valueX>
  - Wert des jeweiligen Attribut

Datensätze löschen:

```
DELETE FROM <table_name> WHERE <attribut_name> = <value>;

mysql> DELETE FROM person_course_execution WHERE fk_participant_id = 7 AND
fk_course_execution_id = 1;
```

- <table_name>
  - Name der Tabelle (gemäss Konventionen)
- <attribute_name>
  - Name des Attributs
- <value>
  - Wert des jeweiligen Attribut

Datenseätze Rekursiv löschen:

```
mysql> DELETE FROM person WHERE id = 7;
--Kursteilneher 'Markus' (id=7) löschen
```

```
→ ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`zli`.`person_course_execution`, CONSTRAINT
`fk_participant` FOREIGN KEY (`fk_participant_id`) REFERENCES `person` (`id`))
```

```
mysql> DELETE FROM person_course_execution WHERE fk_participant_id = 7;
--Löschen aller Kursteilnahmen vom Benutzer Markus (id=7)

mysql> DELETE FROM person WHERE id = 7;
--Kursteilneher 'Markus' (id=7) löschen
```

Datensätze löschen – Rekursiv (ON DELETE CONSTRAINT)

```
mysql> CREATE TABLE person_course_execution (
         id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         fk_participant_id INT UNSIGNED NOT NULL,
         fk_course_execution_id INT UNSIGNED NOT NULL,

         PRIMARY KEY(id),
         FOREIGN KEY(fk_participant_id) REFERENCES person(id) ON DELETE CASCADE,
         FOREIGN KEY(fk_course_execution_id) REFERENCES course_execution(id) ON DELETE CASCADE
);
```

```
mysql> DELETE FROM person WHERE id = 7;
--Kursteilneher 'Markus' (id=7) löschen
```

TCL:

Transaktionen:

BEGIN;

…

COMMIT; oder **ROLLBACK;**

- **COMMIT** – Speichert die Änderungen
- **ROLLBACK** – Verwirft die Änderungen

DCL:

Benutzer und zugriffe - Benutzer erstellen:

```
CREATE USER <username>@'<location>' IDENTIFIED BY '23456';
SET PASSWORD FOR <username>@'<location>' = PASSWORD('12345');
```

```
mysql> CREATE USER backoffice@'localhost' IDENTIFIED BY '23456';
mysql> SET PASSWORD FOR backoffice @'localhost' = PASSWORD('12345');

mysql> CREATE USER backup@'%' IDENTIFIED BY '89765';
```

Benutzer und zugriffe - Benutzer umbenennen:

```
RENAME USER <username>@'<location>' TO <username2>@'<location2>';
```

```
mysql> RENAME USER backoffice@'localhost' TO backoffice2@'localhost';
mysql> RENAME USER backoffice2@'localhost' TO backoffice@'%';
```

Benutzer und zugriffe - Berechtigungen:

```
GRANT <PERMISSION>[,<PERMISSION>] ON <database>.<table> TO <username>@'<location>';
FLUSH PRIVILEGES;
```

```
mysql> GRANT SELECT ON zli.* TO bkp@'localhost';
mysql> GRANT INSERT,UPDATE,DELETE ON zli.course_execution TO backoffice@'localhost';
mysql> GRANT INSERT,UPDATE,DELETE ON zli.person_course_execution TO
backoffice@'localhost';
mysql> FLUSH PRIVILEGES;

REVOKE <PERMISSION>[,<PERMISSION>] ON <database>.<table> FROM <username>@'<location>';
FLUSH PRIVILEGES;
```

```
mysql> REVOKE DELETE ON zli.* FROM office@'localhost';
mysql> FLUSH PRIVILEGES;

SHOW GRANTS FOR <username>@'<location>';
```

```
SHOW GRANTS FOR backoffice@'localhost';
```

Benutzer und zugriffe - Benutzer Löschen:

```
DROP USER <username>@'<location>';
```

```
mysql> DROP USER backoffice@'localhost';
```

DRL:

DATEN abfragen select:

```
SELECT [DISTINCT] { * | Attributliste | mathematische Ausdrücke} Bezeichner
FROM Tabelle1 Bezeichner1, Tabelle2 Bezeichner2, ...
[WHERE Bedingungen]
[GROUP BY Attributliste]
[HAVING Bedingungen]
[ORDER BY Attributliste] [ASC | DESC];
[LIMIT 2(,4)]
```

DATEN abfragen Attribute:

```
SELECT * FROM person;
SELECT id, firstname FROM person;
SELECT id, firstname, lastname, email FROM person;
SELECT id as ID, firstname as Vorname, lastname as Nachname, email as eMail
FROM person;
SELECT id ID, firstname Vorname, lastname Nachname, email eMail FROM person;
```

DATEN abfragen WHERE:

```
SELECT * FROM person WHERE id > 6 AND id < 12;

SELECT * FROM person WHERE id >= 6 AND id <= 12;

SELECT * FROM person WHERE id >= 6 AND id <= 12 AND firstname <> 'Markus';

SELECT * FROM person WHERE id > 20 OR id = 1 OR firstname = 'Markus' OR email
IS NULL;

SELECT * FROM person WHERE (id > 6 AND id < 12) OR MOD(id, 2) = 1;
```

ORDER:

```
SELECT * FROM person WHERE id >= 1 ORDER BY id DESC;

SELECT * FROM person WHERE id >= 1 ORDER BY lastname ASC,
firstname ASC;
```

Funktionen:

```
SELECT id, BIN(id), REVERSE(firstname), UPPER(lastname), email FROM person;
```

→ Referenz für Funktionen

LIKE:

```
SELECT id, firstname, lastname FROM person WHERE firstname LIKE '<PATTERN>';
```

LIKE Patterns:

› **%** - Beliebiges Zeichen, (0..*)

› **_** - Beliebiges Zeichen, (1)

Beispiele:

› `'M%', 'Ma%', '%a%', 'Anna%', '_ _ _ _ _', '_ a _ _ %', '_ _ _ _ ', ' _ _ _ _ _ %', '_a%', '_a%_n'`

LIMIT:

```
SELECT id, firstname, lastname FROM person LIMIT 5;
SELECT id, firstname, lastname FROM person LIMIT 2, 5;
```

Beispiele:

› **LIMIT 5** – Nur 5 Datensätze anzeigen (gemäss Sortierung)

› **LIMIT 2, 5** – Beginnend ab dem 2ten, nur 5 Datensätze (gemäss Sortierung)

Subqueris Value:

```
SELECT * FROM course WHERE price = 200;

SELECT * FROM course WHERE price = (
        SELECT MIN(price) FROM course
);
```

Subqueris Tables:

```
SELECT AVG(price) FROM course LIMIT 3;
```

```
+------------+
| AVG(price) |
+------------+
| 460.000000 |
+------------+
```

```
SELECT AVG(tmp.price) FROM (
        SELECT price FROM course LIMIT 3
) as tmp;
```

```
+----------------+
| AVG(tmp.price) |
+----------------+
|     500.000000 |
+----------------+
```

```
+----+------+--------+
| id | name | price  |
+----+------+--------+
|  1 | 101  | 300.00 |
|  2 | 106  | 500.00 |
|  3 | 114  | 700.00 |
|  4 | 183  | 600.00 |
|  5 | 307  | 200.00 |
+----+------+--------+
```

Gruppen Funktionen:

```
SELECT COUNT(id), MIN(price), AVG(price), MAX(price) FROM course;
```

- **COUNT(…)** – Anzahl der Datensätze
- **MIN(…)** – Minimaler Wert
- **AVG(…)** – Durchschnitts Wert
- **MAX(…)** – Maximaler Wert

→ Gruppenfunktionen geben genau einen Wert zurück. Darum dürfen diese (für den Moment) nicht mit *normalen* Feldern vermischt werden, welche pro Datensatz einen Wert zurückgeben!

INDEXES:

```
EXPLAIN SELECT * FROM babyname WHERE name = 'Markus';
```

```
MariaDB [nation]> explain SELECT * FROM babyname WHERE name = 'Markus';
+------+-------------+----------+------+---------------+------+---------+------+--------+-------------+
| id   | select_type | table    | type | possible_keys | key  | key_len | ref  | rows   | Extra       |
+------+-------------+----------+------+---------------+------+---------+------+--------+-------------+
|    1 | SIMPLE      | babyname | ALL  | NULL          | NULL | NULL    | NULL | 257403 | Using where |
+------+-------------+----------+------+---------------+------+---------+------+--------+-------------+
```

```
CREATE INDEX index_name ON babyname(name);
EXPLAIN SELECT * FROM babyname WHERE name = 'Markus';
```

```
MariaDB [nation]> explain SELECT * FROM babyname WHERE name = 'Markus';
+------+-------------+----------+------+---------------+------------+---------+-------+------+-----------------------
| id   | select_type | table    | type | possible_keys | key        | key_len | ref   | rows | Extra
+------+-------------+----------+------+---------------+------------+---------+-------+------+-----------------------
|    1 | SIMPLE      | babyname | ref  | index_name    | index_name | 303     | const | 47   | Using index condition
+------+-------------+----------+------+---------------+------------+---------+-------+------+-----------------------
1 row in set (0.000 sec)
```

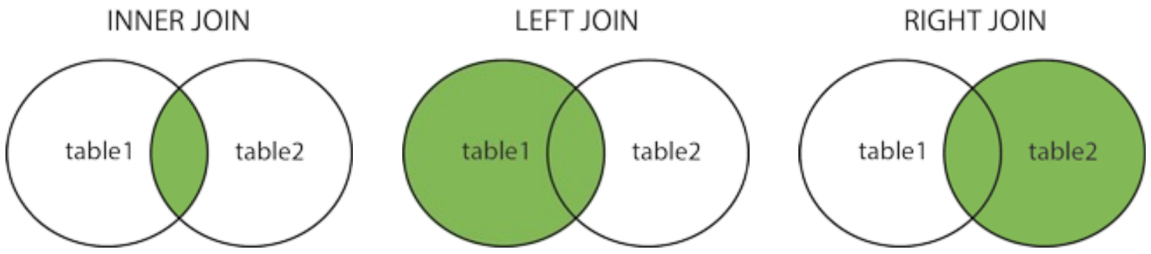
Gruppierung von Daten mit Bedingung:

```
SELECT
        rating,
        MIN(length),
        MAX(length)
FROM film
GROUP BY rating
HAVING MIN(length) > 46;
```

rating	MIN(length)	MAX(length)
G	47	185
R	49	185

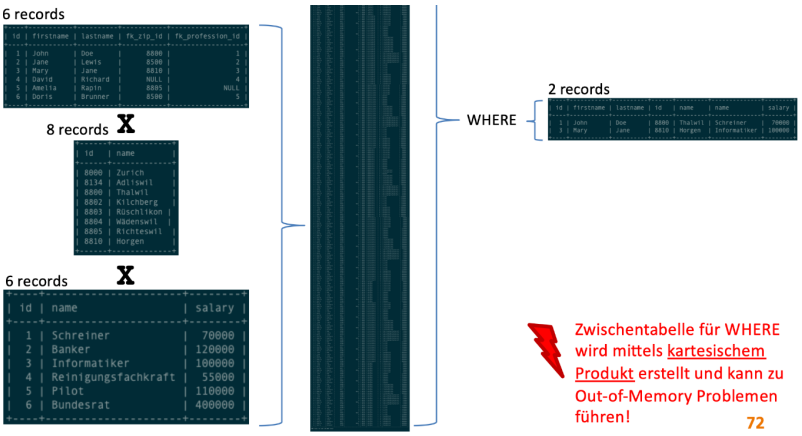
*→ Gruppenfunktionen **MIN**(…) und **MAX**(…) **pro** rating, welches MIN(…) grösse 46 ist. Pendent zum WHERE in einem **GROUP BY**.*

JOINs:



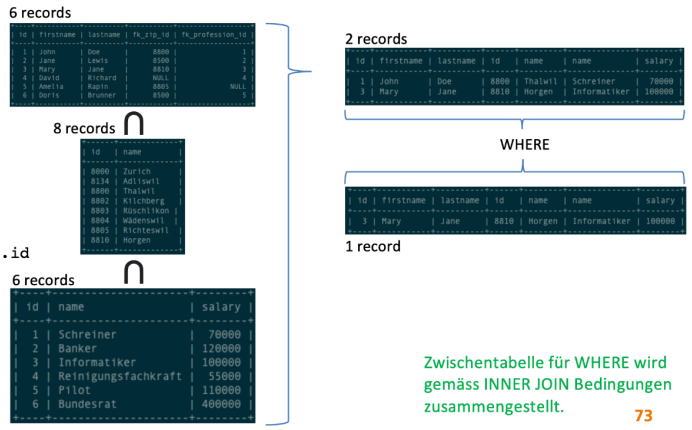
CROSS JOIN:

```
SELECT
        p.id,
        p.firstname,
        p.lastname,
        z.id,
        z.name,
        pr.name,
        pr.salary
FROM
        person p,
        zip z,
        profession pr
WHERE
        p.fk_zip_id = z.id AND
        p.fk_profession_id = pr.id;
```



INNER JOIN:

```
SELECT
   p.id,
   p.firstname,
   p.lastname,
   z.id,
   z.name,
   pr.name,
   pr.salary
FROM
   person p
INNER JOIN zip z ON p. fk_zip_id = z.id
INNER JOIN profession pr ON p.fk_profession_id = pr.id
WHERE
   p.id = 3;
```



Zwischentabelle für WHERE wird gemäss INNER JOIN Bedingungen zusammengestellt.

73

LEFT JOIN:

```
SELECT
   p.id,
   p.firstname,
   p.lastname,
   z.id,
   z.name,
   pr.name,
   pr.salary
FROM
   person p
LEFT JOIN zip z ON p. fk_zip_id = z.id
LEFT JOIN profession pr ON p.fk_profession_id = pr.id
WHERE
   p.id = 3;
```

Zwischentabelle für WHERE wird gemäss LEFT JOIN Bedingungen zusammengestellt.

74

RIGHT JOIN:

```
SELECT
   p.id,
   p.firstname,
   p.lastname,
   z.id,
   z.name,
   pr.name,
   pr.salary
FROM
   person p
RIGHT JOIN zip z ON p. fk_zip_id = z.id
RIGHT JOIN profession pr ON p.fk_profession_id = pr.id
WHERE
   p.id = 3;
```

Import Export:

DUMP:

```
## Dump exportieren
$> mysqldump.exe -u root -p zli > /tmp/zli-kurswesen.sql
$> mysqldump.exe -u root -p --all-databases > /tmp/all-databases.sql

## Dump importieren
mysql> CREATE DATABASE zli_copy;
mysql> exit
$> mysql.exe -u root -p zli_copy < /tmp/zli-kurswesen.sql
```

CSV export:

```
mysql> SELECT
  id, name, description, price, created
FROM
  course
WHERE
  price > 0
INTO OUTFILE '/tmp/zli-courses.csv'
FIELDS
  TERMINATED BY ','
  ENCLOSED BY ''
  LINES TERMINATED BY '\r\n';
```

CSV import:

```
mysql> LOAD DATA INFILE
          '/tmp/zli-courses.csv'
INTO TABLE
          courses
CHARACTER SET
          'latin1'
FIELDS
  TERMINATED BY ','
  ENCLOSED BY ''
  LINES TERMINATED BY '\r\n';
;
```