Inhaltsverzeichnis

[1 Theorie 2](#_Toc518333328)

[1.1 Datenbank 2](#_Toc518333329)

[1.2 Relationale Datenbank 2](#_Toc518333330)

[1.3 Beziehungstypen 2](#_Toc518333331)

[1.4 Datentypen 2](#_Toc518333332)

[1.5 Wertvergleich 2](#_Toc518333333)

[1.6 Zusätze 3](#_Toc518333334)

[1.7 Joker- / Wildcard Zeichen 3](#_Toc518333335)

[1.8 Bedingungen 3](#_Toc518333336)

[1.9 Werte filtern 3](#_Toc518333337)

[1.9.1 SELECT XXX FROM hallo 3](#_Toc518333338)

[1.9.2 SELECT \* FROM hallo XXX 3](#_Toc518333339)

[1.10 Was ist SQL 4](#_Toc518333340)

[1.10.1 SQL-Sprachen 4](#_Toc518333341)

[1.10.2 Sprachgenerationen 4](#_Toc518333342)

[1.11 Normalformen 4](#_Toc518333343)

[1.11.1 Nullte Normalform (Unnormalisierte Form) 4](#_Toc518333344)

[1.11.2 Erste Normalform 5](#_Toc518333345)

[1.11.3 Zweite Normalform 5](#_Toc518333346)

[1.11.4 Dritte Normalform 5](#_Toc518333347)

[2 SQL Commands 6](#_Toc518333348)

[2.1 Datenbanken 6](#_Toc518333349)

[2.2 Tabellen 6](#_Toc518333350)

[2.3 Spalten 7](#_Toc518333351)

[2.3.1 hinzufügen, ändern und löschen 7](#_Toc518333352)

[2.3.2 Primärschlüssel / Sekundärschlüssel 7](#_Toc518333353)

[2.3.3 Fremdschlüssel 7](#_Toc518333354)

[2.4 Tabelleninhalte 7](#_Toc518333355)

[2.5 Indexierung 8](#_Toc518333356)

[2.6 Datenselektion 8](#_Toc518333357)

[2.7 Selektion mit Bedingungen 8](#_Toc518333358)

[2.7.1 Will de Raphi schwanger isch 9](#_Toc518333359)

[2.8 Importieren und Exportieren 9](#_Toc518333360)

[2.9 Join 10](#_Toc518333361)

[2.9.1 cross join 10](#_Toc518333362)

[2.9.2 inner join (join) 10](#_Toc518333363)

[2.9.3 natural join 11](#_Toc518333364)

[2.9.4 left outer join 11](#_Toc518333365)

[2.9.5 right outer join 11](#_Toc518333366)

[2.10 Union 12](#_Toc518333367)

[2.11 Benutzerverwaltung 13](#_Toc518333368)

[2.11.1 Benutzer 13](#_Toc518333369)

[2.11.2 Berechtigungen 13](#_Toc518333370)

# 

# Theorie

## Datenbank

* In einer Datenbank werden Daten abgelegt und in einer vorgegebenen form strukturiert gespeichert.
* Ein Datenbankmanagementsystem (DBMS) ermöglicht die effiziente Einstellung und Verwaltung beziehungsweise Pflege einer Datenbank.

## Relationale Datenbank

Bei einer relationalen Datenbank werden die Informationen in verschieden Tabellen aufbewahrt. Über sogenannte Schlüssel können diese Tabellen und damit deren Datensätze beliebig miteinander verbunden werden. (Relationen)

Primarschlüssel ist eine eindeutige Identifikationsnummer zum Beispiel UR 36

Der Fremdschlüssel ist der Primärschlüssel eines anderen Datensatzes

## Beziehungstypen

## Datentypen

**char(10)** // reserviert genau 10 Zeichen

**varchar(10)** // max Eingabe 10 Zeichen; kann auch weniger sein – maximal 255

**text** // für mehr als 255 Zeichen

**numeric** // Zahlen mit oder ohne Kommastellen

**float** // Zahlen mit Kommastellen

**integer** // Zahlen ohne Kommastellen

**blob** // Binar large object (maximal ca. 4GB)

**date** // 2018-06-26

**time** // 13:37:42

**timestamp** // 2018-06-26 13:37:42

## Wertvergleich

**=** //gleich

**<** // kleiner als

**>** // grösser als

**<=** // kleiner oder gleich

**>=** // grösser oder gleich

**<>** // ungleich

## Zusätze

**not null** // Zelle muss einen Wert enthalten

**default** // falls kein Wert eingetragen wird, wird der default wert eingefügt.

**check** // damit kann man Werte einschränken – **check (PLZ > 0)**

**auto\_increment** // wird null eingetragen, zählt es automatisch hoch

## Joker- / Wildcard Zeichen

**\_** // genau ein Zeichen // **#**

**%** // beliebiges Zeichen (keines, eines oder viele) // **\***

"Me\_er" // Meier, Meyer

"Me%er" // Meier, Meyer, Mehlbauer

## Bedingungen

|  |  |  |
| --- | --- | --- |
| SELECT | Was | **PLZ** |
| FROM | Woher | **hallo** |
| WHERE | Bedingung | **vorname = lxbauzz** |
| HAVING | Bedingung | **nachname** |
| ORDER BY | Sortierung | **nachname = Gertsch** |

## Werte filtern

### SELECT XXX FROM hallo

**\*** // zeigt alle Spalten

**pers\_id, nachname** // zeigt Spalten pers\_id und nachname

**pers\_id AS PersonenID** // gibt der Spalte pers\_id den Alias PersonenID

**DISTINCT nachname** // unterdrückt die gleiche Ausgabe von zwei gleichen Nachnamen

**COUNT(\*)** // zählt die Anzahl Datensätze der Spalte in der Klammer

**AVG (PLZ)** // zählt den Mittelwert der Spalte PLZ

**SUM (PLZ)** // zählt alle Werte zusammen von der Spalte PLZ

**MIN (PLZ)** // gibt den kleinsten Wert der Spalte PLZ

**MAX (PLZ)** // gibt den grössten Wert der Spalte PLZ

### SELECT \* FROM hallo XXX

**WHERE nachname LIKE|= "Gertsch"** // zeigt alle Datensätze mit dem Nachnamen Gertsch

**WHERE PLZ (NOT) BETWEEN 3000 AND 4000** // zeigt Datensätze zwischen 3000 und 4000

**WHERE PLZ IS (NOT) NULL** // Ausgabe von Datensätzen welche teils (nicht) leer sind

**ORDER BY nachname ASC|DESC** // sortiert die Ausgabe der Datensätze nach Nachnamen

**GROUP BY nachname** // gruppiert die Werte nach Nachnamen

## Was ist SQL

### SQL-Sprachen

**Data Definition Language (DDL)**

Damit kann man Datenbanken, Tabellen und Attribute erstellen und bearbeiten (CREATE, ALTER, DROP)

**Data Control Language (DCL)**

Mit dieser Sprache bearbeitet man die Rechte der User. (GRANT, REVOKE)

**Data Manipulation Language (DML)**

Hiermit fügt man Inhalte in eine Tabelle und kann diese ändern (INSERT, UPDATE, DELETE)

**Data Query Language (DQL)**

Diese Sprache dient nur zur Auswertung der Daten. (SELECT)

### Sprachgenerationen

**1. Generation** (Maschinensprache)

Alle Programme wurden nur mit 0 und 1 programmiert. Erste Kurzbefehle wurden programmiert: Beispielsweise ADD für Addition und JMP um an einen anderen Ort im Programm zu gelangen.

**2. Generation** (Assembler)

Eingabe von Kurzwörtern kann dank Assembler ein Computer selber in 0 und 1 umwandeln. Nur für einen bestimmten Prozessor geeignet. Soll das Programm auf einen anderen Prozessor übertragen werden, muss es neu geschrieben werden.

**3. Generation** (höhere prozedurale Programmiersprachen)

Quelltext wird in einer leicht verständlichen Sprache geschrieben. Unabhängig von Prozessoren. Man muss genau erklären was man erreichen will und wie es erreicht werden kann.

Beispiele sind BASIC, C, JAVA, PHP

**4. Generation** (nicht prozedurale Programmiersprachen)

Der Programmierer braucht nur noch anzugeben, was er erreichen will.

|  |  |
| --- | --- |
| **3. Generation** | **4. Generation** |
| 1. Lies alle Namen aus Tabelle "Kunden"  2. Führe nächste Schritte so lange aus, bis alle Datensätze durch sind.  3. Lies nächsten Datensatz  4. Falls Name = "Meier", zeige ihn mit den anderen zusammen an  5. Gehe zu Schritt 2 | 1. Zeige mir alles aus Tabelle "Kunden" mit dem Namen "Meier" |

## Normalformen

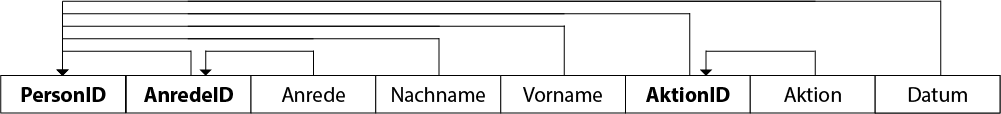
### Nullte Normalform (Unnormalisierte Form)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PersonID** | **Anrede** | **Nachname** | **Vorname** | **Aktion** |
| 1 | Herr | Gertsch | Marcel | 24.06.2018 01 Discobesuch  25.06.2018 02 Kinobesuch  26.06.2018 03 Schulbesuch |
| 2 | Frau | Sommer | Anika | 24.06.2018 02 Kinobesuch  26.06.2018 02 Kinobesuch |

### Erste Normalform

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PersonID** | **Anrede** | **Nachname** | **Vorname** | **AktionID** | **Aktion** | **Datum** |
| 1 | Herr | Gertsch | Marcel | 01 | Discobesuch | 24.06.2018 |
| 1 | Herr | Gertsch | Marcel | 02 | Kinobesuch | 25.06.2018 |
| 1 | Herr | Gertsch | Marcel | 03 | Schulbesuch | 26.06.2018 |
| 2 | Frau | Sommer | Anika | 02 | Kinobesuch | 24.06.2018 |
| 2 | Frau | Sommer | Anika | 02 | Kinobesuch | 26.06.2018 |

### Zweite Normalform



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***PersonID*** | **AnredeID** | **Anrede** | **Nachname** | **Vorname** |
| 1 | H | Herr | Gertsch | Marcel |
| 2 | F | Frau | Sommer | Anika |
| 3 | H | Herr | Schillig | Nemo |

|  |  |
| --- | --- |
| ***AktionID*** | **Aktion** |
| 01 | Discobesuch |
| 02 | Kinobesuch |
| 03 | Schulbesuch |

|  |  |  |  |
| --- | --- | --- | --- |
| **PersonID** | **AktionID** | **Aktion** | **Datum** |
| 1 | 01 | Discobesuch | 24.06.2018 |
| 1 | 02 | Kinobesuch | 25.06.2018 |
| 1 | 03 | Schulbesuch | 26.06.2018 |
| 2 | 02 | Kinobesuch | 24.06.2018 |
| 2 | 02 | Kinobesuch | 26.06.2018 |
| 3 | 03 | Schulbesuch | 31.07.2018 |

### Dritte Normalform

|  |  |  |  |
| --- | --- | --- | --- |
| **PersonID** | ***AnredeID*** | **Nachname** | **Vorname** |
| 1 | H | Gertsch | Marcel |
| 2 | F | Sommer | Anika |
| 3 | H | Schillig | Nemo |

|  |  |
| --- | --- |
| **AktionID** | **Aktion** |
| 01 | Discobesuch |
| 02 | Kinobesuch |
| 03 | Schulbesuch |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***PersonID*** | ***AktionID*** | **Datum** |  | **AnredeID** | **Anrede** |
| 1 | 01 | 24.06.2018 |  | H | Herr |
| 1 | 02 | 25.06.2018 |  | F | Frau |
| 1 | 03 | 26.06.2018 |  |
| 2 | 02 | 24.06.2018 |  |
| 2 | 02 | 26.06.2018 |  |
| 3 | 03 | 31.07.2018 |  |

# SQL Commands

## Datenbanken

**CREATE DATABASE IF NOT EXIST elite;**

// erstellt eine Datenbank "elite", wenn diese noch nicht existiert

**SHOW DATABASES;**

// zeigt alle Datenbanken

**USE elite;**

// wechseln zur Datenbank "elite"

**DROP DATABASE elite;**

// löscht die Datenbank "elite"

## Tabellen

**SHOW TABLES;**

// zeigt alle Tabellen in der aktuellen Datenbank an

**CREATE TABLE hallo (**

**pers\_id INTEGER NOT NULL AUTO\_INCREMENT,**

**anrede CHAR (4) DEFAULT "Herr",**

**nachname VARCHAR (55) NOT NULL,**

**vorname VARCHAR (55),**

**PLZ INTEGER DEFAULT 1111,**

**EntryDate TIMESTAMP,**

**PRIMARY KEY (pers\_id),**

**FOREIGN KEY (PLZ) REFERENCES Orte (PLZ)**

**);**

// pers\_id als integer Attribut, darf nicht leer sein, automatisch hochzählen

// anrede als Zeichen Attribut, Max 4 Zeichen, Standardgemäss "Herr"

// nachname als Zeichen Attribut, Max 55 Zeichen, darf nicht leer sein

// vorname als Zeichen Attribut, Max 55 Zeichen

// PLZ als integer Attribut

// EntryDate als Zeitformat YYYY-MM-DD hh:mm:ss

// pers\_id wird als Primärschlüssel festgelegt

// PLZ wird als Fremdschlüssel festgelegt, referenzierend aus Tabelle "Orte", Attribut "PLZ"

**DROP TABLE hallo;**

// löscht Tabelle hallo unwiderruflich samt Inhalt

**DESCRIBE hallo;**

// zeigt die Spalten mit allen Eigenschaften von hallo

## Spalten

### hinzufügen, ändern und löschen

**ALTER TABLE hallo ADD TelNr INTEGER;**

// fügt die Spalte TelNr in Tabelle hallo

**ALTER TABLE hallo CHANGE TelNr TelefonNr INTEGER;**

// überschreibt in der Tabelle hallo die Spalte TelNr mit TelefonNr

**ALTER TABLE hallo DROP TelNr;**

// löscht Spalte TelNr

### Primärschlüssel / Sekundärschlüssel

**ALTER TABLE hallo ADD PRIMARY KEY (pers\_id);**

// fügt im Nachhinein einen Primärschlüssel ein

**ALTER TABLE hallo DROP PRIMARY KEY (pers\_id);**

// entfernt einen Primärschlüssel

**ALTER TABLE hallo ADD UNIQUE (PLZ);**

// definiert PLZ als Sekundärschlüssel

### Fremdschlüssel

**ALTER TABLE hallo ADD FOREIGN KEY (PLZ) REFERENCES orte (PLZ);**

// fügt im Nachhinein einen Fremdschlüssel ein

**ALTER TABLE hallo DROP FOREIGN KEY PLZ;**

// entfernt einen Fremdschlüssel

## Tabelleninhalte

#### hinzufügen, löschen und ändern

**INSERT INTO hallo**

**(pers\_id, anrede, nachname, vorname, PLZ, TelNr, EntryDate)**

**VALUES (null, "Herr" "Gertsch", "Marcel", 3084, 0792131999, current\_timestamp);**

// fügt alle lila Werte in die zugewiesenen Attribute. Attribute können weggelassen werden, wenn alle VALUES eingesetzt werden. Beispiel:

**INSERT INTO hallo VALUES (null, "Herr", "Gertsch", "Marcel", 3084, 0792131999, current\_timestamp);**

// current\_timestamp fügt das aktuelle Datum +Uhrzeit ein

**INSERT INTO hallo2 (pers\_id, nachname, vorname)**

**SELECT pers\_id, nachname, vorname FROM hallo;**

// dies kopiert alle Inhalte von hallo in die Tabelle hallo2

**TRUNCATE TABLE hallo;**

// löscht alle Inhalte unwiderruflich aus dieser Tabelle

**DELETE FROM hallo WHERE nachname LIKE | = "Gertsch";**

// löscht den gesamten Datensatz, bei welchem Nachname "Gertsch" ist.

**UPDATE hallo SET nachname="Gertsch" WHERE pers\_id LIKE | = 2;**

// ändert in der Spalte bei der pers\_id = 2 ist den Nachnamen zu "Gertsch"

**UPDATE hallo SET nachname="Gertsch", vorname="Marcel" WHERE pers\_id=2 AND PLZ=3084;**

## Indexierung

**CREATE INDEX** **idx\_nachname ON** **hallo (Nachname);**

// erstellt einen Index namens "idx\_nachname" für die den Attribut Nachname.

**ALTER TABLE hallo ADD INDEX idx\_nachname (Nachname);**

// fügt einen Index hinzu

**ALTER TABLE hallo DROP INDEX idx\_nachname;**

// entfernt und löscht den Index

## Datenselektion

**SELECT \* FROM hallo;**

// zeigt alle Spalten mit deren Inhalt von hallo

**SELECT nachname FROM hallo;**

// zeigt nur die Spalte Nachname mit all deren Zeilen

**SELECT pers\_id AS PersonenID FROM hallo;**

// pers\_id erhält den Alias PersonenID

**SELECT pers\_id AS PersonenID, vorname AS Name FROM hallo;**

// pers\_id erhält den Alias PersonenID, Vorname erhält den Alias Name

## Selektion mit Bedingungen

**SELECT \* FROM hallo WHERE vorname = "lx%" AND PLZ = 3084;**

// es werden nur Datensätze dargestellt, bei welchen diese Bedingungen zutreffen

**SELECT \* FROM hallo WHERE PLZ >= 4000;**

// selektiert jeden Datensatz, bei welchem PLZ grosser oder gleich wie 4000 ist.

**SELECT \* FROM hallo WHERE PLZ BETWEEN 3000 AND 4000;**

// selektiert jeden Datensatz, bei welchem PLZ zwischen 3000 und 4000 ist.

**SELECT \* FROM hallo WHERE PLZ IS NULL;**

// selektiert jeden Datensatz, bei welchem PLZ leer ist.

**SELECT nachname, vorname FROM hallo ORDER BY vorname, nachname;**

// nachname und vorname wird alphabetisch nach vorname geordnet

**SELECT \* FROM hallo ORDER BY pers\_id ASC|DESC;**

// sortiert Tabelle nach pers\_id

**SELECT DISTINCT PLZ FROM hallo;**

// unterdrückt die gleiche Ausgabe von zwei gleichen Postleitzahlen

**SELECT COUNT(PLZ), MIN(PLZ), MAX(PLZ), SUM(PLZ), AVG(PLZ), FROM hallo;**

// zählt alle PLZ Einträge, kleinster & höchster Wert, Summe und Mittelwert von PLZ.

,

**SELECT PLZ, COUNT(PLZ) AS Anzahl FROM hallo GROUP BY PLZ;**

// gibt die Anzahl von PLZ Sätzen aus der Tabelle hallo welche nach PLZ gruppiert werden

**SELECT CONCAT (nachname, vorname) FROM hallo;**

// hängt Strings aneinander ohne Trennung

**SELECT CONCAT\_WS (", " nachname, vorname) FROM hallo;**

// hängt String aneinander mit ", " Trennung

***SUBSTRING (STRING, STARTZEICHEN [LÄNGE])***

**SELECT nachname, SUBSTRING (vorname, 1, 1) FROM hallo;**

// Substring nimmt einen String auseinander

**SELECT nachname, SUBSTRING (vorname, 1, 1) AS "Initialen" FROM hallo;**

// nimmt den ganzen nachnamen, 1. Zeichen von vorname und deklariert das als Initialen

**SELECT CONCAT(SUBSTRING(VORNAME,1,1), SUBSTRING(NACHNAME,1,1)) AS INITIALEN FROM hallo;**

// zeigt 1. Buchstaben von vorname und den 1. Buchstaben von nachname

**SELECT COUNT(vorname) as TotalVorname FROM hallo;**

//zählt alle Datensätze, bei welchem der vorname ausgefüllt wurde

// WICHTIG, NACH COUNT KEIN LEERZEICHEN!

### Will de Raphi schwanger isch

**SELECT DISTINCT(product\_category) FROM product ORDER BY product\_category;**

// Zeigen Sie alle unterschiedlichen Produktkategorien an und sortieren Sie die Ausgabe aufsteigend

**SELECT DISTINCT(country) FROM customer;**

// Zeigen Sie alle unterschiedlichen Herkunftsländer der Kundentabelle an

**SELECT product\_category, COUNT(\*) AS n\_products FROM product GROUP BY product\_category;**

// Geben Sie die Anzahl der Produkte je Produktkategorie aus

**SELECT country, SUM(salary) AS sum\_salary FROM employee WHERE country="Switzerland";**

// Geben Sie die Summe der Gehälter der Mitarbeiter aus der Schweiz an

**SELECT product\_category, COUNT(\*) AS n\_products, SUM(product\_price) AS sum\_price**

**FROM product GROUP BY product\_category;**

// Geben Sie die Anzahl der Produkte und die Summe der Preise je Produktkategorie aus

**SELECT product\_category, COUNT(\*) AS n\_products, SUM(product\_price) AS sum\_price**

**FROM product GROUP BY product\_category;**

// Geben Sie die Anzahl der Produkte und die Summe der Preise je Produktkategorie

// sortiert nach Anzahl Produkte aus.

## Importieren und Exportieren

**LOAD DATA INFILE "Pfad" INTO TABLE hallo;**

// importiert Daten von einer externen Datei in eine Tabelle (Beispieldatei: .csv)

**SOURCE Pfad**

// importiert Datenbank von einer externen Datei

// Datei muss bei XAMPP bereits in "C:/xampp/mysql/data" sein

**mysql -p < /Users/lxbauzz/Desktop/elite.sql**

// importieren externen Dateien in die Datenbank elite

**mysqldump -u root -p elite > /Users/lxbauzz/Desktop/elite.sql**

// im **Terminal** eingegeben, exportiert es die Datenbank elite in die Datei elite.sql

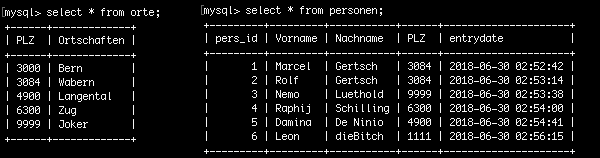
WICHTIG: ES WERDEN NUR TABELLEN EXPORTIERT. Spielt eine Rolle beim Importieren.

**mysqldump -u root -p --databases elite > /Users/lxbauzz/Desktop/elite.sql**

WICHTIG: ES WERDEN ALLE TABELLEN MIT DER DATENBANK EXPORTIERT

## Join

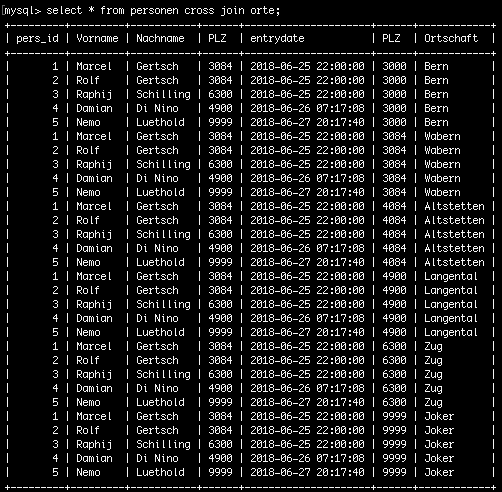
Joins sind dafür da, Tabellen zu verknüpfen. Die zweite Normalform ist das beste Beispiel, warum man Joins braucht. Um bei der Eingabe/Ausgabe nicht verwirrt zu sein, da Spalten teilweise den gleichen Namen haben, immer Tabelle.Spalte eingeben. (personen.plz, orte.plz)

Unsere Beispiel Tabellen sehen so aus:

### cross join

Verbindet jede Zeile von personen mit jeder Zeile von orte. Cross join ist die unwichtigste Variante.

**SELECT \* FROM personen CROSS JOIN orte;**

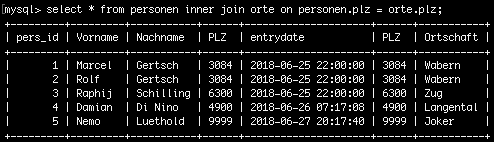
Das selbe ohne cross join: **SELECT \* FROM personen, orte;**

### inner join (join)

Überprüft anhand der "on" Bedingung, ob bei der erst angegeben Spalte ein Wert von der zweit angegebenen Spalte übereinstimmt und gibt dies dann zusammen als ein Datensatz aus. Wie man im Beispiel sieht, werden die Tabellen nur hintereinander gehändt. PLZ ist doppelt vorhanden. Ist ein Wert in personen vorhanden, jedoch nicht in orte, dann wird dieser komplett ignoriert.

**SELECT \* FROM personen INNER JOIN orte on personen.PLZ = orte.PLZ;**

Das selbe ohne join: **SELECT \* FROM personen, orte WHERE personen.PLZ = orte.PLZ;**



### natural join

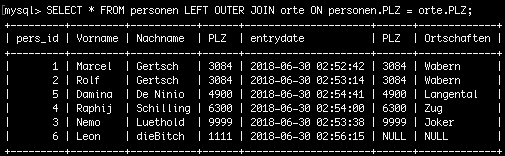
die "intelligentere" Variante von inner join. Natural join überprüft automatisch, welche Spalten doppelt vorhanden sind. Es löscht eine der doppelten Spalten, die andere setzt es an den Anfang. Auch hier werden Datensätze ignoriert, wenn PLZ in Tabelle orte nicht zugewiesen wurde.

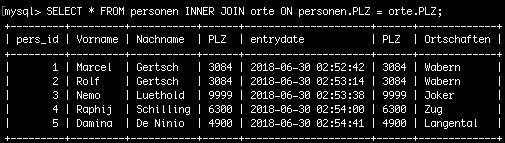
**SELECT \* FROM personen NATURAL JOIN orte;**



### left outer join

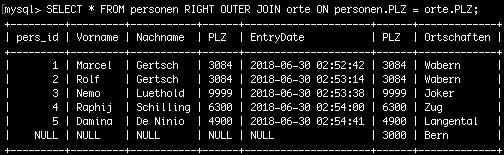
Der Unterschied zum inner join: hierbei werden Datensätze, welche nicht genau übereinstimmen nicht ignoriert, sondern es werden die leeren Felder mit null gefüllt.

**SELECT \* FROM personen LEFT OUTER JOIN orte ON personen.PLZ = orte.PLZ;**

****Nochmals zum Vergleich: **SELECT \* FROM personen INNER JOIN orte on personen.PLZ = orte.PLZ;**

### right outer join

Macht genau dasselbe wie left outer join, nur beachtet es die Daten von der zweit genannten Tabelle.

**SELECT \* FROM personen RIGHT OUTER JOIN orte ON personen.PLZ = orte.PLZ;**

## Union

**SELECT anrede, nachname, vorname, strasse, PLZ FROM personenraphi**

**UNION**

**SELECT anrede, nachname, vorname, strasse, PLZ FROM personenmarcel;**

// Die Abfrage der ersten Tabelle (Raphi) wird mit der Tabelle (Marcel) verbunden. Es werden nur Daten gezeigt, welche doppelt vorkommen.

**INSERT INTO tbleinladung (anrede, nachname, vorname, strasse, PLZ)**

**SELECT anrede, nachname, vorname, strasse, PLZ FROM personenraphi**

**UNION**

**SELECT anrede, nachname, vorname, strasse, PLZ FROM personenmarcel;**

// speichert die gemeinsamen Kontakte in eine gemeinsame Tabelle ab

Verknüpfte Abfrage ausführen

**SELECT tbleinladung.anrede,** // anzuzeigende Attribute mit ihren Herkunfttabelle

**tbleinladung.nachname, tbleinladung.vorname,**

**tbleinladung.strasse, tbleinladung.PLZ, tblor.ort**

**FROM tbleinladung, tblOrt** // Es werden alle notwendigen Tabellen aufgeführt

**WHERE tbleinladung.PLZ = tblort.PLZ;**

// die übereinstimmenden Datenfelder (Primär und Fremdschlüssel) der Tabelle angegeben werden.

Ersatzname für Tabellen

**SELECT a.anrede, a.nachname, a.vorname, a.strasse, a.PLZ, b.ort**

**FROM tbleinladung AS a, tblOrt as b**

**WHERE a.PLZ = b.PLZ;**

// hierfür müssen Sie lediglich nach Form die Tabellennamen mit as umbennen

Verknüpfte Abfrage mit Bedingungen ausführen

**SELECT a.anrede, a.nachname, a.vorname, a.strasse, a.PLZ, b.ort**

**FROM tbleinladung a, tblOrt as b**

**WHERE a.PLZ = b.PLZ AND b. ort = "Züri";**

// verknüpfte Abfragen können immer mit zusätzlichen Bedingung kombiniert werden

Gemeinsame Datensätze ermitteln

**SELECT a.anrede,**

**a.nachname,**

**a.vorname,**

**a.strasse,**

**a.PLZ**

**FROM tblperson\_isabella AS a, tblperson\_georg AS b**

**WHERE a.nachname = b.nachname**

**AND a.vorname = b.vorname;**

// Definition von Ersatznamen von Tabellen + Gemeinamen Datensatz ermitteln

## Benutzerverwaltung

### Benutzer

**SELECT user, host FROM mysql.user;**

// zeigt Benutzer und deren host von der Tabelle user, ohne die Datenbank wechseln zu müssen

**CREATE USER lxbauzz identified by "misPW";**

// erstellt einen Benutzer mit dem Passwort "misPW"

**SET PASSWORD FOR lxbauzz = "neuesPW";**

// aktualisiert das Passwort von lxbauzz

**DROP USER lxbauzz;**

// löscht Benutzer lxbauzz

**mysql -h localhost -u lxbauzz -p**

// User lxbauzz meldet sich im Bereich localhost an

### Berechtigungen

**GRANT ALL ON \*.\* TO lxbauzz;**

// dem Benutzer lxbauzz werden alle Rechte zugewiesen.

**GRANT SELECT ON elite.hallo TO lxbauzz;**

// berechtigt lxbauzz den SELECT Befehl in der Tabelle hallo zu benutzen

**REVOKE SELECT ON** **elite.hallo FROM** **lxbauzz;**

// enzieht die Berechtigungen, dass lxbauzz den SELECT Befehl in der Tabelle hallo benutzen darf

**SHOW GRANTS FOR lxbauzz;**

// zeigt alle Berechtigungen, die der User lxbauzz hat.

**Folgende Rechte sind zu vergeben:**

**ALL** // gewährt alle Rechte

**SELECT** // gewährt Leserechte, also das Recht zur Ausführung der Abfragen

**UPDATE** // gewährt Rechte zum Ändern der Datensätze

**DELETE** // gewährt Rechte zum Löschen der Datensätze

**REFERENCES** // gewährt Rechte zum Definieren von Fremdschlüsseln

**ALTER** // gewährt Rechte um Tabellen und Attribute zu verändern **DROP** // gewährt Rechte zum Entfernen von Attributen oder Schlüsseln

**CREATE** // gewährt Rechte, Datenbanken und Tabellen zu erstellen

**INDEX** // gewährt Rechte, Index hinzuzufügen