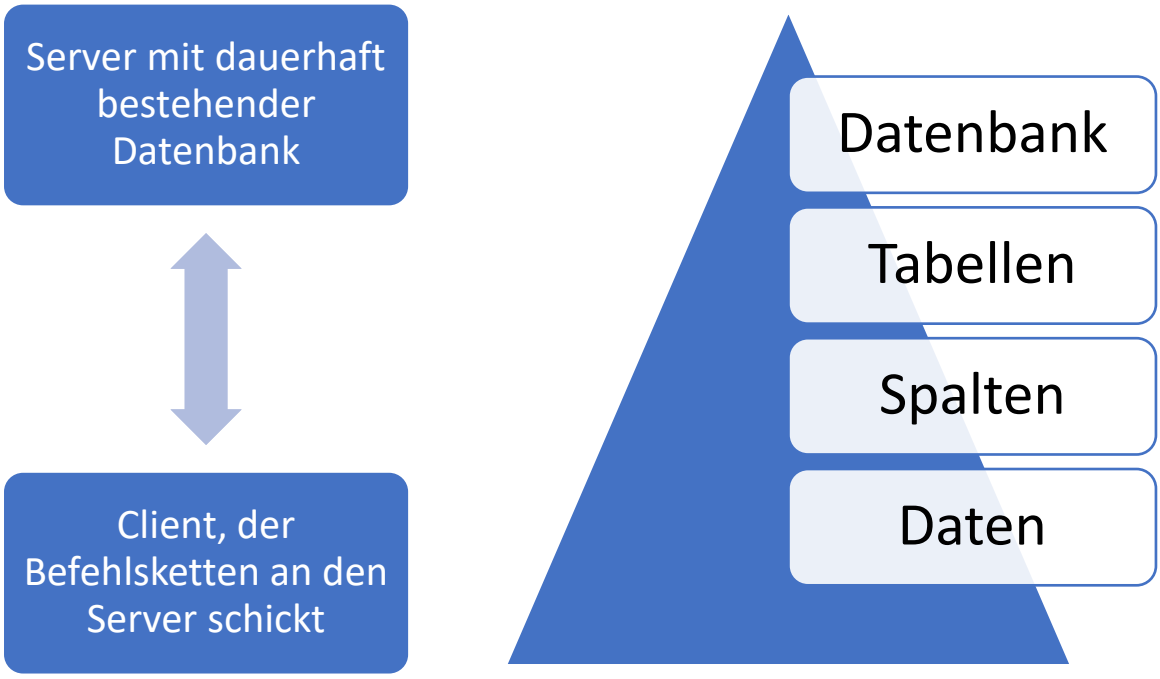
Datenbanken mit SQL

Wiki

Wie ist eine Datenbank aufgebaut?



Wichtig zu verstehen ist, dass die **Datenbank nach dem Ausführen** aller Befehle immernoch **weiter besteht**. (Es sei denn, sie wurde per Befehl gelöscht.) Aus diesem Grund muss man, wenn man ein und dieselbe Befehlskette mehrmals ausführen will, die Datenbank zu Beginn jedes Mal wieder löschen.

Eine Datenbank ist im Grund nichts weiter als eine **Sammlung verschiedener Tabellen**. Dazu haben diese allerdings die Eigenschaft, durch Programme wie SQL feste Befehle und Möglichkeiten zur Verfügung zu haben, um die Tabellen zu **managen** und miteinander zu **verknüpfen**. Letzendlich sind Datenbanken meistens dazu ausgelegt, auch riesige Datenmengen enthalten zu können. Beispielsweise Amazon verwendet eine riesige Zentraldatenbank aller Produkte. Nutzer und Händler greifen dann darauf zu und ändern sie ggf.

Informationen wie Primärschlüssel und Fremdschlüssel werden auf diesem Lernzettel nicht erläutert, da sie in der vorangegangenen Klausur bereits abgefragt wurden.

SQL-Sprachelemente

Data Definition Language (DDL)
Definition von Datenbanken, Tabellen und anderen Strukturen.

Data Control Language (DCL)
Zugriffsberechtigungen.

Data Manipulation Language (DML)
Manipulation und Abfrage der Daten selbst.

Datentypen

Name	Deklaration	Beispiele	Erklärung
Integer	int(Stellen)	0, 481, 18, -23	Ganze Zahlen
VarChar	varchar(Stellen)	„Penis“, „31e“	Zeichen oder Zahlen. Muss in " " oder `` deklariert werden.
Decimal	decimal(Stellen, Dezimalstellen)	9.2 , 18.223	Dezimalzahl, Kommatas werden als Punkte geschrieben.
Date	date()	2018-05-17	Datum. Format: YYYY-MM-DD
Year	year()	1999, 2022	Jahre.
Float	float(Stellen, Dezimalstellen)	3.381 , 3.01	Kleine Dezimalzahl mit vielen Stellen hinterm Komma. Kommatas werden als Punkte geschrieben.

Primärschlüssel

Beim Erstellen von Primärschlüsseln innerhalb des Datenbankerstellungsbefehls, gilt folgendes:

<Name> <Datentyp> **NOT NULL AUTO_INCREMENT**,
PRIMARY KEY (<Namen>)

Also zum Beispiel:

s_*nr int(4)* **not null auto_increment**,
primary key*(s_nr)*

NOT NULL bedeutet, dass das Tabellenfeld niemals leer sein darf. NULL bedeutet also nicht 0 sondern nichts.

AUTO_INCREMENT bedeutet, dass (falls nicht angegeben) automatisch der nächst höhere Primärschlüssel generiert wird.

Nach der Erstellungsklammer muss **engine = innodb;** stehen. SQL bietet verschiedene Speichersysteme, die die Daten letzendlich verwalten. Innodb ist eine kostenlose Variante davon.

Fremdschlüssel

Beim Erstellen von Fremdschlüsseln innerhalb des Datenbankerstellungsbefehls, gilt folgendes:

<Fremdschlüsselname> <Datentyp>,
foreign key(<Fremdschlüsselname>) **references** <Tabellenname>(<Primärschlüssel>)

klasse_nr varchar(10),
foreign key(klasse_nr) **references** Klasse(klasse_nr)

Zusätzlich zu der Namensbeziehung, kann man bei Fremdschlüsseln angeben, was beim Löschen oder Ändern der Referenz passiert. Man nennt dies Referentielle Integrität.

Prefix: ON DELETE, ON UPDATE

Suffix:

- **RESTRICT**: Es wird verhindert, dass die Referenz gelöscht oder aktualisiert wird.
- **CASCADE**: Löschen/Ändern wird einfach übernommen.
- **SET NULL**: Datensatz wird geleert, bleibt aber bestehen.
- **NO ACTION**: Es passiert nichts.

Funktionen

Diese Funktionen können statt einem Spaltenname im Selectbefehl verwendet werden.

SELECT **Avg**() FROM Tabelle;	|| Berechnet den Durchschnitt einer Spalte. Avg(Spalte)
SELECT **Sum**() FROM Tabelle;	|| Berechnet die Summe einer Spalte. Sum(Spalte)
SELECT **Max**() FROM Tabelle;	|| Zeigt den größten Wert einer Spalte an. Max(Spalte)
SELECT **Min**() FROM Tabelle;	|| Zeigt den kleinsten Wert einer Spalte an. Min(Spalte)
SELECT **Count**() FROM Tabelle;	|| Zeigt die Menge der Felder einer Spalte. Count(Spalte)
[…] **ORDER BY** Spalte;	|| Sortiert die Ergebnisse von A-Z oder 0-9.
[…] **ORDER BY** Spalte **DESC**;	|| Sortiert die Ergebnisse absteigend, also von Z-A oder 9-0.
WHERE Spalte=Wert;	|| Gibt nur die Ergebnisse aus, wo der Wert übereinstimmt.
WHERE Spalte>Wert;	|| Siehe oben. Möglichkeiten: > oder < oder >= oder <=

Gruppierungen

Es ist möglich, bestimmte Abfragen in Gruppen zusammenzufassen. Dazu verwendet man den **GROUP BY** Befehl. Das hat den Effekt, dass man zum Beispiel, wenn man nach Jahr gruppiert, immer nur ein Film pro Jahr ausgegeben wird. Nützlich ist das, wenn man zum aus einer Tabelle alle Städte und deren Einwohner aus mehreren Bezirken sehen möchte. Dann gruppiert man nach dem Stadtnamen, um jede Stadt nur einmal zu nennen, und nutzt die sum() Funktion für die Einwohnerspalte.

Außerdem möglich ist es, die Gruppe zusätzlich mit Bedingungen einzuschränken. Dazu verwendet man folgenden Befehl: **GROUP BY** <Spalte> **HAVING [Bedingung]**

Inner Join

Der Inner Join ist eine Verknüpfung zweier Tabellen, bei der ein Wert aus beiden Tabellen identisch ist. Nützlich ist das, wenn Daten aus zwei Tabellen, die beispielsweise über eine m:n Beziehung verbunden sind, ausgegeben werden sollen. Das Funktioniert folgendermaßen:

```
SELECT Bahnhof FROM DeutscheBahn
  INNER JOIN Busbahnhof ON Bahnhof.p_nr=Busbahnhof.f_nr WHERE Bedingung;
```

Es können auch mehr als zwei Tabellen verknüpft werden, indem weitere INNER JOIN Befehlsteile angehängt werden und jeweils mit der vorherigen Tabelle verknüpft werden. Beispiel:

```
SELECT Bahnhof FROM DeutscheBahn
  INNER JOIN Busbahnhof ON Bahnhof.p_nr=Busbahnhof.f_nr WHERE Bedingung
  INNER JOIN Taxibahnhof ON Busbahnhof.p_nr=Taxibahnhof.f_nr WHERE Bedingung;
```

Commands

Legende: COMMAND <Erforderlich> [Optional]

Syntax	Funktion
CREATE DATABASE <Datenbankname>;	Erstellt die Datenbank.
USE <Datenbankname>;	Wählt die Datenbank aus. *Man kann auch mit mehreren parallel arbeiten*
DROP DATABASE <Datenbankname>	Löscht die Datenbank.
CREATE TABLE <Tabellenname> (<Spaltenname> <Datentyp> [Optionen], [**PRIMARY KEY** (<Schlüsselname>)] [**FOREIGN KEY** (<Schlüsselname>) **REFERENCES** <Tabellenname>(Primärschlüsselname)]) engine = innodb;	Erstellt die Tabelle. Ordnet wahlweise Primär und Fremdschlüssel zu.
DROP TABLE <Tabellenname>	Löscht die Tabelle.
INSERT INTO <Tabellenname> (<Spalten>) **VALUES** (<Werte>);	Fügt Daten in eine Tabelle ein.
SELECT [DISTINCT] <Spaltennamen> **FROM** <Tabellenname(n)> **[Where-Bedingung];**	Wählt bestimmte Datensätze aus, um diese anzuzeigen. Funktioniert auch über mehrere Tabellen hinweg.
ALTER TABLE <Tabellenname> **CHANGE** <Alter Name> <Neuer Name> <Datentyp>;	Ändert den Namen einer Spalte und/oder dessen Datentyp.
ALTER TABLE <Tabellenname> **DROP** <Spaltennamen>;	Löscht eine Spalte aus einer Tabelle.
ALTER TABLE <Tabellenname> **ADD** <Spaltenname> <Datentyp>;	Fügt eine neue Spalte hinzu.
DELETE FROM <Tabellenname> **WHERE** <Bedingung>**;**	Löscht alle Datensätze, auf die die Bedingung zutrifft.
UPDATE <Tabellenname> **SET** <Spaltenname>=<Neuer Wert> **WHERE** [Bedingung]	Ändert alle Datensätze in einer Spalte, auf die die Bedingung zutrifft.
SELECT <Spaltennamen> **FROM** <Tabelle1> **INNER JOIN** <Tabelle2> **ON** [Verknüpfung];	Fügt eine zusätzliche Tabelle an die Auswahl an am Punkt einer Verknüpfung.
SELECT <Spaltennamen> **FROM** <Tabellenname> [Where-Bedingung] **GROUP BY** <Spaltenname> [**HAVING** <Bedingung>];	Gruppiert die Abfrage zusätzlich. Diese Gruppierung kann zusätzlich mit einer Bedingung verknüpft werden, indem **HAVING** verwendet wird.