# INHALTSVERZEICHNIS

# ABBILDUNGSVERZEICHNIS

# ABKÜRZUNGSVERZEICHNIS

## C

CLI
Call Level Interface ..... 6, 7

## D

DBMS
Datenbankmanagementsystem ..... 6, 10
DDL
Data Definition Language ..... 7
DML
Data Manipulation Language ..... 7, 8

## E

ER
Entity-Relationship ..... 35

## I

ID
Identifier ..... 47, 52, 53, 56, 58, 59, 63, 72, 76, 80, 83, 87

## P

php
Perl Hypertext Preprocessor ..... 17, 33, 34
PSM
Persistent Stored Modules ..... 6, 7

## S

SQL
Structured Query Language ..... 6, 7, 9, 10, 17, 18, 37, 87

## X

Xampp
X Apache MySQL PHP Perl ..... 11, 17, 18, 24

# 1 KONZEPT

## 1.1 EINFÜHRUNG

Die Abfragesprache SQL ist die etablierte Sprache für die Arbeit mit relationalen Datenbankmanagementsystemen (DBMS). SQL ist die Sprache, mit der die meisten relationalen Datenbanken erstellt, manipuliert und abgefragt werden**.** Mit einer geeigneten Abfragesprache können gewünschte Daten in einer relationalen Datenbank gesucht werden. Es existieren verschiedene Standards, und jeder Hersteller von DBMS hat seine eigenen Erweiterungen und Besonderheiten zu den Standards.
Die erste Version der **SQL-Norm** wurde **1986** veröffentlicht. SQL war nicht als vollständige Programmiersprache entworfen worden, sondern als Abfrage-Sprache.
1989 wurde eine Revision, die **Norm SQL:1989**, veröffentlicht. Diese Version erweitert SQL:1986 um den Begriff der referentiellen Integrität[1].
Eine wesentliche Revision erschien 1992 als **SQL:1992**. Diese brachte Erweiterungen in fast allen Teilen der Sprache. Darunter fallen zusätzliche Datentypen, OUTER JOINs, Kataloge, Domänen, Zuweisungen, temporäre Tabellen, referentielle Aktionen, eine Schema Manipulation Language, dynamisches SQL, Information Schema Tables sowie eine größere Orthogonalität der Sprache selbst.

- Die erste inkrementelle **Teil-Norm** erschien **1995**, ein neues Call Level Interface (SQL/CLI). CLI funktioniert als aufrufbare Schnittstelle zu einem SQL-Datenbank-System und stellt so eine hohe Dynamik bereit, ganz im Gegensatz zu den relativ statischen Eigenschaften von eingebundenem SQL.
- Die zweite inkrementelle **Teil-Norm**, SQL/PSM (Persistent Stored Modules), erschien **1996**. Die Norm ermöglicht es, Anwendungs-Logik aus den Anwendungsprogrammen in das Datenbank-System zu verlagern.

1999 gab es eine weitere Revision der Norm, **SQL:1999**. Diese erweitert die Idee der inkrementellen Teile und schafft damit eine fünfteilige Norm, die aus folgenden Teilen besteht:

- Teil 1: SQL/Framework
- Teil 2: SQL/Foundation
- Teil 3: SQL/CLI
- Teil 4: SQL/PSM und
- Teil 5: SQL/Bindings

[1] referentielle Integrität

In **Teil 1:** SQL/Framework wird die Beziehung zwischen den verschiedenen Teilen beschrieben. Zusätzlich werden Begriffe, Definitionen und Conformance-Anweisungen[2] beschrieben, die für alle Teile gelten. **Teil 2:** SQL/Foundation ist der umfangreichste Teil. Er enthält alle DDL[3]- und DML[4]-Elemente von SQL:1992 (allerdings ohne die Spezifikationen für *embedded* und *dynamic* SQL), die in SQL aufrufbaren Routinen von SQL/PSM sowie viele neue Sprachmerkmale, die seit der Veröffentlichung von SQL:1992 entwickelt wurden. Für die SQL:1999-Versionen sowohl von **Teil 3:** SQL/CLI als auch von **Teil 4:** SQL/PSM wurden die entsprechenden Versionen von 1995 bzw. 1996 geringfügig überarbeitet. **Teil 5:** SQL/Bindings enthält die Spezifikationen für *embedded* und *dynamic* SQL von SQL-92, entsprechend überarbeitet, wobei die neuen Merkmale aus SQL/Foundation berücksichtigt wurden.

Die Revision des Jahres 2003 enthält den gesamten Funktionsumfang von SQL:1999, aber auch einen neuen Teil. Neue Sprachelemente in **SQL:2003** sind:

**MERGE-Anweisung**

Führt Einfüge-, Update- oder Löschvorgänge in einer Zieltabelle anhand der Ergebnisse eines JOINs mit einer Quelltabelle aus. Es können z. B. zwei Tabellen synchronisiert werden, indem die Zeilen in einer Tabelle anhand von Unterschieden, die in der anderen Tabelle gefunden wurden, eingefügt, aktualisiert oder gelöscht werden.
Die Merge-Anweisung steht für eine Kombination von mehreren Update- und Insert-Anweisungen, die ausgewählte Spaltenwerte bzw. Zeilen aus einer Tabelle in eine andere Tabelle übernimmt:

```
MERGE INTO <Zieltabellenname>
[AS <Korrelationsname>]
USING <Tabellenreferenz>
ON <Verbundbedingung>
WHEN
[MATCHED THEN SET <Spaltenzuweisung>]
[NOT MATCHED THEN
INSERT [(<Spalten>)] VALUES (<Werte>)]
```

Mindestens eine der beiden Matched-Klauseln muss angegeben werden. Je nachdem, ob die Verbundbedingung erfüllt ist oder nicht, wird die jeweilige Klausel ausgeführt.

[2] Conformance-Anweisungen
[3] Datenbeschreibungssprache
[4] Datenverarbeitungssprache

**Identitätsspalten**

Eine Identitätsspalte ermöglicht die automatische Generierung von Schlüsseln mit Hilfe eines impliziten Sequenzgenerators. Der Typ einer solchen Spalte wird folgendermaßen angegeben:

```
GENERATED {ALWAYS | BY DEFAULT}
AS IDENTITY (<Sequenzoptionen>)
```

ALWAYS schließt ein Update der Spalte aus. BY DEFAULT dagegen erlaubt Inserts und Updates.

**Generierte Spalten (abgeleitete Daten)**

Eine generierte Spalte ist eine Spalte, deren Wert aus Werten anderer Spalten derselben Zeile berechnet wird. Eine generierte Spalte wird wie folgt definiert:

```
<Spaltenname> GENERATED ALWAYS
(<Wertausdruck>)
```

Die Variablen im Wertausdruck sind auf nicht generierte Spalten derselben Zeile sowie Funktionen beschränkt, die weder SQL-Anfragen noch DML-Anweisungen enthalten.

**Sequenzgeneratoren**

Ein Sequenzgenerator erzeugt Sequenzen von numerischen Werten, z.B. für die Vergabe künstlicher Schlüsselwerte:

```
CREATE SEQUENCE <Sequenzname>
AS <Typname>
[START WITH <Wert>]
[INCREMENT BY <Wert>]
[NO MINVALUE | MINVALUE <Wert>]
[NO MAXVALUE | MAXVALUE <Wert>]
[NO CYCLE | CYCLE]
```

Die Angabe der Sequenzoptionen erfolgt reihenfolgeunabhängig. Ist der Inkrementwert negativ, erfolgt eine Dekrementierung. Wird beim Inkrementieren der Maximalwert (bzw. beim Dekrementieren der Minimalwert) erreicht, entscheidet die CYCLE-Klausel über den nächsten Schritt. Ist NO CYCLE definiert, wird die Generierung mit dem Setzen einer Ausnahmebedingung beendet. CYCLE dagegen setzt die Generierung mit dem Minimalwert fort. Der Zugriff auf den nächsten Wert einer Sequenz erfolgt mit dem Ausdruck NEXT VALUE FOR <Sequenzname>.

**TABLESAMPLE**

Stichprobenverfahren werden oft benutzt, um den Aufwand zu vermeiden, eine vollständige Grundgesamtheit (zum Beispiel eine umfangreiche Tabelle) zu analysieren. Die neue TABLESAMPLE-Klausel, die als Tabellenreferenz in der FROM-Klausel auftritt, erlaubt eine zufällige Auswahl von Zeilen. Die statistische Analyse einer eingeschränkten Ergebnismenge kann wesentlich schneller beendet werden und trotzdem alle Informationen liefern, die für Aufgaben wie Entscheidungsfindung benötigt werden.

**Mehrfach-Zuweisungen**

Zuweisungen werden in SQL zum einen dazu verwendet, in UPDATE- und MERGE-Anweisungen jeder Zeile, die von der Anweisung betroffen ist, die entsprechenden Spaltenwerte zuzuweisen, zum anderen, um lokale Variablen in Anwendungsprogrammen, Stored procedure [5] und benutzerdefinierten Funktionen mit Werten zu besetzen. In SQL:1999 ist diese Zuweisung nur von „einfacher“ Art, das heißt, das Ergebnis der Auswertung eines einzelnen skalaren Ausdrucks kann nur einem einzelnen Ziel zugewiesen werden. Mehrfach-Zuweisungen erlauben es, jedes einzelne Feld eines n-Tupels mit skalaren Werten dem entsprechenden Element in einer Liste von Zielvariablen zuzuweisen. Die skalaren Ausdrücke, die das n-Tupel bilden, werden dabei alle ausgewertet, bevor irgendeine Zuweisung stattfindet. Bekannt ist dieses Konzept bereits von FETCH- und SELECT INTO-Anweisungen, nun ist es auch in regulären Zuweisungen verfügbar.

**MULTISET-Typen**

Eine Multimenge ist eine Sammlung gleichgetypter Objekte, ähnlich wie eine Menge, ohne bestimmte Ordnung, jedoch mit Duplikaten. So ist z.B. {1,2,3,4} ≠ {1,2,2,3,3,4} da die Kardinalität nicht übereinstimmt. MULTISET erstellt eine Instanz einer Multimenge aus einer Liste von Werten. Leere Multimengenkonstruktoren sind nicht zulässig.

```
MULTISET (expression [{ ,expression}])
OR
{expression [{ ,expression}]}
```

**Ausblick**

Es lässt sich mit einiger Sicherheit vorhersagen, dass man auch in absehbarer Zukunft an den SQL-Normen weiterarbeiten wird. Die Datenbankhersteller müssen sich auch in Zukunft großen Herausforderungen an die Funktionalität und Leistungsfähigkeit ihrer Produkte stellen, was wiederum die Fortentwicklung der Normen vorantreiben wird. Die SQL-Normen haben sich etabliert als ausgereifte Normen mit breiter Akzeptanz, aber auch als Normen, die immer noch in der Lage sind, neue Technologien aufzunehmen.

[5] gespeicherte Prozedur

Diese Anleitung soll eine Einführung in die Sprache SQL bieten. Ziel ist es, dass nach dem Durcharbeiten folgende Aufgaben selbständig gelöst werden können:

- Eigene einfache relationale Datenbank aufbauen
- Datenbank mit MySQL Workbench verknüpfen
- Abfragen für relationale Datenbanken durchführen

Um die Ziele zu erreichen, wird SQL anhand praxisnaher Beispiele erläutert. In dieser Anleitung wird im Tool phpMyAdmin eine Datenbank angelegt und mit MySQL Workbench vereinigt. MySQL Workbench hält sich an **SQL** 99 und kennt nur ausgewählte Elemente von **SQL** 2003. Vorzugsweise werden allgemeingültige Schreibweisen nach dem SQL-Standard benutzt. Deshalb sollten die Befehle in aller Regel auf allen gängigen DBMS funktionieren und höchstens kleinere Änderungen benötigen. Dort, wo eine spezielle Schreibweise wesentlich abweicht, wird das ausdrücklich erwähnt.

## 1.2 Motivation der verwendeten Methodik

| Kapitel 2 | Datenmodell „Kundenauftragsverwaltung“ |
|---|---|
| **Woher?**<br>Ausgangspunkt<br>Problemstellung | Zum Erlernen und Üben der SELECT-Befehle fehlt eine standardisierte, didaktisch geeignete Arbeitsumgebung für Studierende. |
| **Wohin?**<br>Zieldefinition | Das Ziel dieses Kapitels ist es, eine solche Arbeitsumgebung zu schaffen. |
| **Wie?**<br>Methoden<br>(-verwendung) | Ein einfaches, didaktisch geeignetes Datenmodell und eine Arbeitsumgebung, in der das Datenmodell implementiert wird und in der SQL-Abfragen möglich sind, werden erstellt. Für die Erläuterung im Selbststudium soll die Verwendung der Arbeitsumgebung erklärt werden. |
| **Was?**<br>Resultate | Zunächst wird ein Kundenauftragsmodell in vier Notationen entwickelt. Die Min-Max- und die Charles-Bachman-Notationen werden mit Visio abgebildet. Die Krähenfuß-Notation wird mit MySQL Workbench und die phpMyAdmin-Notation mit XAMPP phpMyAdmin erstellt.<br>Zur Implementierung dieses Datenmodells wird eine Datenbank eingerichtet, die dann mit MySQL Workbench verbunden wird. In phpMyAdmin werden die Tabellen angelegt und mit MySQL Workbench verknüpft. Eine Verknüpfung mit MySQL Workbench ist zwar nicht erforderlich, wird in dieser Arbeit aber zusätzlich angeboten, um den Umgang mit beiden Tools vertraut zu machen. |

| Kapitel 3 | WHERE-Klausel |
|---|---|
| **Woher?**<br>Ausgangspunkt<br>Problemstellung | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung der WHERE-Klausel erlernen und üben können. |
| **Wohin?**<br>Zieldefinition | Es soll ein Abschnitt einer Anleitung erstellt werden, der die Problemstellung löst. |
| **Wie?**<br>Methoden<br>(-verwendung) | Die WHERE-Klausel ist theoretisch und anhand von Beispielen zu erklären. Übungsaufgaben für eine bestimmte Anzahl wesentlicher Befehlsvarianten sind zu entwerfen. |
| **Was?**<br>Resultate | Dieser Abschnitt der Anleitung enthält die Befehlsvarianten:<br>=, !=, <, >, <=, >=, BETWEEN und LIKE.<br>Die jeweiligen Varianten werden theoretisch erklärt, und dazu wird je ein Beispiel mit Kommentierung angegeben. Für die Studierenden sind zwei bis vier Übungsbeispiele angegeben, die sie selbständig lösen und kommentieren sollen. |

| Kapitel 4 | JOIN |
|---|---|
| **Woher?**<br>Ausgangspunkt<br>Problemstellung | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des JOINs erlernen und üben können. |
| **Wohin?**<br>Zieldefinition | Es soll ein Abschnitt einer Anleitung erstellt werden, der die Problemstellung löst. |
| **Wie?**<br>Methoden<br>(-verwendung) | Der JOIN ist theoretisch und anhand von Beispielen zu erklären. Übungsaufgaben für eine bestimmte Anzahl wesentlicher Befehlsvarianten sind zu entwerfen. |
| **Was?**<br>Resultate | Dieser Abschnitt der Anleitung enthält die JOIN-Varianten:<br>INNER JOIN, LEFT OUTER JOIN, NEGATIVER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, CROSS JOIN und SELF-JOIN.<br>Die jeweiligen Varianten werden theoretisch erklärt, und dazu wird je ein Beispiel mit Kommentierung angegeben. Für die Studierenden sind zwei Übungsbeispiele angegeben, die sie selbständig lösen und kommentieren sollen. |

| Kapitel 5 | Bezug zur Mengenlehre und Logik |
|---|---|
| **Woher?**<br>Ausgangspunkt<br>Problemstellung | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung von Mengenlehre und Logik in Bezug auf SQL erlernen und üben können. |
| **Wohin?**<br>Zieldefinition | Es soll ein Abschnitt einer Anleitung erstellt werden, der die Problemstellung löst. |
| **Wie?**<br>Methoden<br>(-verwendung) | Mengenlehre und Logik sind theoretisch und anhand von Beispielen zu erklären. Übungsaufgaben für eine bestimmte Anzahl wesentlicher Befehlsvarianten sind zu entwerfen. |
| **Was?**<br>Resultate | Dieser Abschnitt der Anleitung enthält die Mengenlehre- und Logik-Konzepte: $\wedge$, $\vee$, \, $\neg$, $\Delta$.<br>Die jeweiligen Konzepte werden theoretisch erklärt, und dazu wird je ein Beispiel mit Kommentierung angegeben. Für die Studierenden sind zwei Übungsbeispiele angegeben, die sie selbständig lösen und kommentieren sollen. |

| Kapitel 6 | Gruppenbildung und Aggregatfunktionen |
|---|---|
| **Woher?**<br>Ausgangspunkt<br>Problemstellung | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung der Gruppenbildung und Aggregatfunktionen erlernen und üben können. |
| **Wohin?**<br>Zieldefinition | Es soll ein Abschnitt einer Anleitung erstellt werden, der die Problemstellung löst. |
| **Wie?**<br>Methoden<br>(-verwendung) | Gruppenbildung und Aggregatfunktionen sind theoretisch und anhand von Beispielen zu erklären. Übungsaufgaben für eine bestimmte Anzahl wesentlicher Befehlsvarianten sind zu entwerfen. |
| **Was?**<br>Resultate | Dieser Abschnitt der Anleitung enthält die Gruppen- und Aggregatfunktions-Varianten: SUM, MIN, MAX, AVG, COUNT.<br>Die jeweiligen Varianten werden theoretisch erklärt, und dazu wird je ein Beispiel mit Kommentierung angegeben. Für die Studierenden sind zwei bis vier Übungsbeispiele angegeben, die sie selbständig lösen und kommentieren sollen. |

| Kapitel 7 | Schwierige Aufgaben |
|---|---|
| **Woher?**<br>Ausgangspunkt<br>Problemstellung | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium echt schwierige Aufgaben erlernen und üben können. |
| **Wohin?**<br>Zieldefinition | Es soll ein Abschnitt einer Anleitung erstellt werden, der die Problemstellung löst. |
| **Wie?**<br>Methoden<br>(-verwendung) | Es wird ein neues komplexes Datenmodell erstellt. Dazu sind schwierige Aufgaben zu definieren, die Themen der anderen Kapitel vertiefen. |
| **Was?**<br>Resultate | Das neue Datenmodell „Vorlesung“ wurde mit Hilfe von Visio erstellt. Dieser Abschnitt der Anleitung enthält schwierigere Aufgaben als in den anderen Kapiteln. Für die Studierenden sind Übungsaufgaben angegeben, die sie selbständig lösen und kommentieren sollen. |

## 1.3 AUFGABENSTELLUNG UND VORGEHENSWEISE

Im Rahmen dieser Bachelorarbeit wurde eine ***„SQL-Anleitung“*** erstellt. Mittels dieser Anleitung können SQL-Abfragen an eine Datenbank gestellt werden.
Als Grundlage für die Durchführung der Abfragen wird die Datenbank in der freien Software XAMPP phpMyAdmin verwendet. Da die Struktur des Aufbaus einer Datenbank nicht umständlich ist, außerdem das Tool zur grafischen Darstellung verwendet werden kann und leicht mit MySQL Workbench zu verbinden ist, wird **phpMyAdmin** [6] gewählt. PhpMyAdmin ist eine freie PHP-Applikation zur Administration von MySQL-Datenbanken. Die Administration erfolgt über HTTP mit einem Browser. Daher können auch Datenbanken auf fremden Rechnern über eine Netzwerkverbindung oder über das Internet administriert werden. Tabellen werden angelegt und anschließend mit Beispieldaten beladen.

Die Frage, wie die graphische Benutzerschnittstelle zu gestalten ist, ist ein Hauptthema, denn jeder Studierende soll möglichst ohne Einarbeitungszeit oder langes Lesen einer Anleitung sofort Abfragen eintippen können. Auch ist wichtig, dass die Studierenden MySQL so nutzen können, dass sie eine grafische Abbildung der Tabellen angezeigt bekommen und dass die SQL-Befehle funktionieren. Es wird angenommen, dass die primären Benutzer die Studierenden des Fachs Datenbanken sein würden.

Besondere Aufmerksamkeit galt auch der Frage, wie die Studierenden langsam an die Aufgaben herangeführt werden sollen, ohne sie dabei ins kalte Wasser zu werfen. Deshalb wird zu jedem der Kapitel eine Beispielaufgabe mit einer Lösung angegeben und dazu mindestens zwei Übungsaufgaben für die Studierenden formuliert, die sie zusätzlich kommentieren sollen. Zusätzlich stellte sich die Frage, welche SQL-Abfragen vorgestellt und den Studierenden beigebracht werden sollten.

Diese Anleitung soll Studierenden im 4. Semester im Fach Datenbanken als Arbeitsumgebung und Übungsvorlage dienen und ihnen helfen, sich gezielt auf die Klausur vorzubereiten.

Ausgangspunkt für das **2. Kapitel** war, dass eine standardisierte Arbeitsumgebung für Studierende fehlte. In diesem Kapitel dieser Anleitung wird das Datenmodell „Kundenauftragsverwaltung“ in vier verschiedenen Notationen gezeigt.
Davor wird eine Datenbank auf dem Server benötigt. In phpMyAdmin können Tabellen angelegt und mit MySQL Workbench verknüpft werden. Eine Verbindung mit MySQL Workbench ist zwar nicht erforderlich, da im Tool phpMyAdmin SQL-Abfragen ausgeführt werden können. Trotzdem wird in dieser Anleitung MySQL Workbench zusätzlich angeboten, um den Studierenden zu zeigen, wie

[6] Kapitel 2.2.1.2 phpMyAdmin – Datenbank anlegen

MySQL Workbench aus bereits existierenden Datenbanken Strukturen herausfindet und ER-Modelle erstellt. Darüber hinaus konvertiert MySQL Workbench Tabellen vom SQL Server in MySQL-Tabellen. Die Min-Max- und die Charles-Bachman-Notation werden mit Visio dargestellt. Die phpMyAdmin-Notation wird mit phpMyAdmin XAMPP und die Krähenfuß-Notation mit MySQL Workbench erstellt.

Das **3. Kapitel** beschreibt die WHERE-Klausel zunächst theoretisch und enthält ein Beispiel je Vergleichsprädikat mit Kommentierung. Zu den jeweiligen Vergleichsprädikaten werden Übungsaufgaben gestellt, die von den Studierenden gelöst und kommentiert werden sollen. Es gibt folgende Vergleichsprädikate, die in dieser Anleitung verwendet werden: gleich, ungleich, größergleich, kleinergleich, größer als, kleiner als, BETWEEN AND und LIKE. Das Ziel ist es, Studierenden die WHERE-Klausel beizubringen.

Die tabellenübergreifende Verknüpfung (JOIN) mit den verschiedenen Varianten wird in **Kapitel 4** beschrieben. Studierende sollen lernen, den JOIN anzuwenden. Dafür werden die verschiedenen Varianten theoretisch erklärt und an je einem Beispiel mit Kommentierung näher erläutert. Dazu werden je Variante Übungsaufgaben definiert, die zu lösen und zu kommentieren sind.

Viele Studierende haben Probleme, Mengenlehre und Logik zu verstehen. In **Kapitel 5** werden Mengenlehre und Logik zuerst erklärt, bevor die verschiedenen Junktoren in SQL-Aufgaben formuliert, erläutert und kommentiert werden. Nach der beispielhaften Erläuterung werden Übungsaufgaben definiert, die gelöst und kommentiert werden sollen. Das Ziel ist es, Verständnisprobleme zu beseitigen.

Im **6. Kapitel** sollen Studierende lernen, Gruppenauswertungen und Aggregatfunktionen durchzuführen. Nach Standard-SQL sind diese die Funktionen SUM (Summieren), MIN (Minimum), MAX (Maximum), AVG (Durchschnitt) und COUNT (Zählen). Diese Funktionen werden zunächst theoretisch erklärt und an einem Beispiel vorgeführt. Dazu werden Übungsaufgaben gestellt, die gelöst und kommentiert werden sollen.

Das anschließende **Kapitel 7** befasst sich mit schwierigeren Aufgaben als in den anderen Kapiteln. Dazu wird ein neues Datenmodell **„Vorlesung“** mit Visio erstellt.
Studierende haben bisher noch keine echt schwierigen Aufgaben gesehen und sollen sich anhand dieses Kapitels auf die Klausur vorbereiten. Das Ziel ist es, selbständig ohne Beispiele an die Aufgaben heranzugehen.

| | Kapitel 2.1 | Kapitel 2.2 | Kapitel 2.2.1 |
|---|---|---|---|
| **Woher?**<br>**Ausgangspunkt Problemstellung** | Zur Durchführung der SQL-Abfragen fehlt ein übersichtliches Datenmodell, welches in ein Tool integriert werden soll. | Es fehlt eine Beschreibung einer Arbeitsumgebung, mit der Studierende eine Datenbank anlegen, SQL-Befehle durchführen und Datenmodelle abbilden können. | Es fehlt eine Beschreibung eines Tools, mit dem Studierende eine Datenbank anlegen, SQL-Befehle durchführen und Datenmodelle abbilden können. |
| **Wohin?**<br>**Zieldefinition** | Ziel ist es, ein solches Datenmodell zu konstruieren. | Das Ziel dieses Abschnitts ist es, eine Installationsanleitung für eine Arbeitsumgebung zu erstellen. | Das Ziel dieses Abschnitts ist es, eine Installationsanleitung für ein solches Tool zu erstellen. |
| **Wie?**<br>**Methoden (-verwendung)** | Es werden vier Notationen verwendet. Die Charles-Bachman-Notation, MIN-/MAX-Notation, phpMyAdmin -und Krähenfuß-Notation. | Die freie Software XAMPP und das Tool MySQL Workbench werden beschrieben und installiert, um den Studierenden den Umgang mit der Datenbank und den SQL-Abfragen vertraut zu machen. | Um das Datenmodell in einer Datenbank abzubilden, wird die Software XAMPP beschrieben und installiert, welche das Tool phpMyAdmin beinhaltet. |
| **Was?**<br>**Resultate** | Das Datenmodell wird in Charles-Bachman- und MIN-MAX-Notation mit Microsoft Visio erstellt. Die phpMyAdmin-Notation wird mit XAMPP phpMyAdmin und die Krähenfuß-Notation mit MySQL Workbench erstellt. | Anleitung für:<br>Mit dem Tool phpMyAdmin, welches in der Software XAMPP vorhanden ist, wird eine Datenbank angelegt, SQL-Befehle ausgeführt und Datenmodelle abgebildet. Zusätzlich wird MySQL Workbench installiert, um bereits existierende Strukturen aus der Datenbank herauszufinden und SQL-Befehle auch mit diesem Tool auszuführen. | Es wird Schritt für Schritt erklärt, wie die Software XAMPP installiert wird. In dem Tool phpMyAdmin wird die Datenbank angelegt und die Tabellen werden eingepflegt. |

| | Kapitel 2.2.2 | Kapitel 2.3 |
|---|---|---|
| **Woher?**<br>**Ausgangspunkt**<br>**Problemstellung** | Es fehlt eine Beschreibung eines Tools, das mit der in phpMyAdmin erstellten Datenbank verbunden werden kann. | Es fehlen Werte in der Datenbank, mit denen SQL-Abfragen durchgeführt werden können. |
| **Wohin?**<br>**Zieldefinition** | Es soll eine Installationsanleitung für ein solches Tool erstellt werden. | Testdaten sollen in die Datenbank eingegeben werden können. |
| **Wie?**<br>**Methoden (-verwendung)** | Es wird beschrieben, wie mit MySQL Workbench eine Verbindung zur Datenbank aufgebaut wird. | Nach der Installation von XAMPP können im Tool phpMyAdmin die Tabellen erstellt und die Werte (Inhalte) eingetragen werden. |
| **Was?**<br>**Resultate** | Anleitung für:<br>MySQL Workbench wird installiert und eine Verbindung zur Datenbank hergestellt.<br>Somit findet MySQL Workbench aus der bereits existierenden Datenbank Strukturen heraus, erstellt ER-Modelle (Krähenfuß-Notation) und übernimmt die Tabellen aus der Datenbank. | In phpMyAdmin ist die Datenbank erstellt, die Struktur der Tabellen festgelegt und die Werte sind eingetragen. |

## Charles-Bachman-Notation:

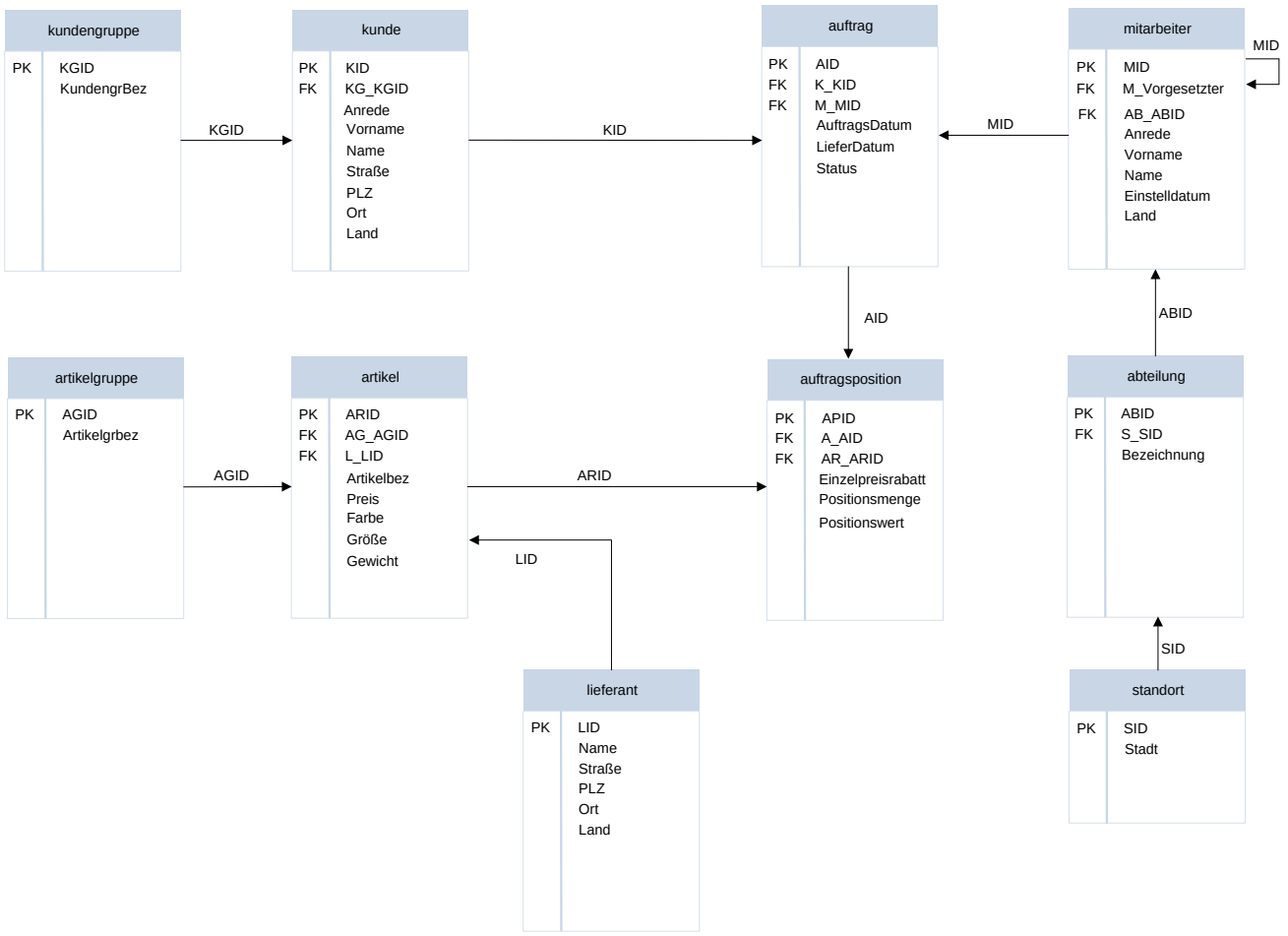


**Abbildung 1: Charles-Bachman-Notation**

### Kommentar zur referentiellen Integrität:

Pfeil bezeichnet 1:n-Beziehung: Die Tabelle an der Pfeilspitze enthält den Primärschlüssel der Tabelle am Pfeilschaft als Fremdschlüssel. Jede Tabelle besitzt einen Primärschlüssel.

Die Tabelle **Kunde** enthält den Fremdschlüssel aus der Tabelle Kundengruppe.

Die Tabelle **Auftrag** enthält die beiden Fremdschlüssel aus den Tabellen Kunde und Mitarbeiter.

Die Tabelle **Mitarbeiter** besitzt den Fremdschlüssel aus der Tabelle Abteilung. Außerdem hat sie einen Fremdschlüssel für den Vorgesetzten aus der eigenen Tabelle, da der Vorgesetzte auch gleichzeitig ein Mitarbeiter ist.

Die Tabelle **Abteilung** besitzt einen Fremdschlüssel aus der Tabelle Standort.

Die Tabelle **Auftragsposition** hat die beiden Fremdschlüssel aus den Tabellen Auftrag und Artikel.

Die Tabelle **Artikel** besitzt die beiden Fremdschlüssel aus den Tabellen Lieferant und Artikelgruppe.

Neben den Primär- und Fremdschlüsseln hat jede Tabelle eigene Attribute.
Die Tabellennamen sind klein und die Attribute der Tabellen sind großgeschrieben.

## Min-Max-Notation:

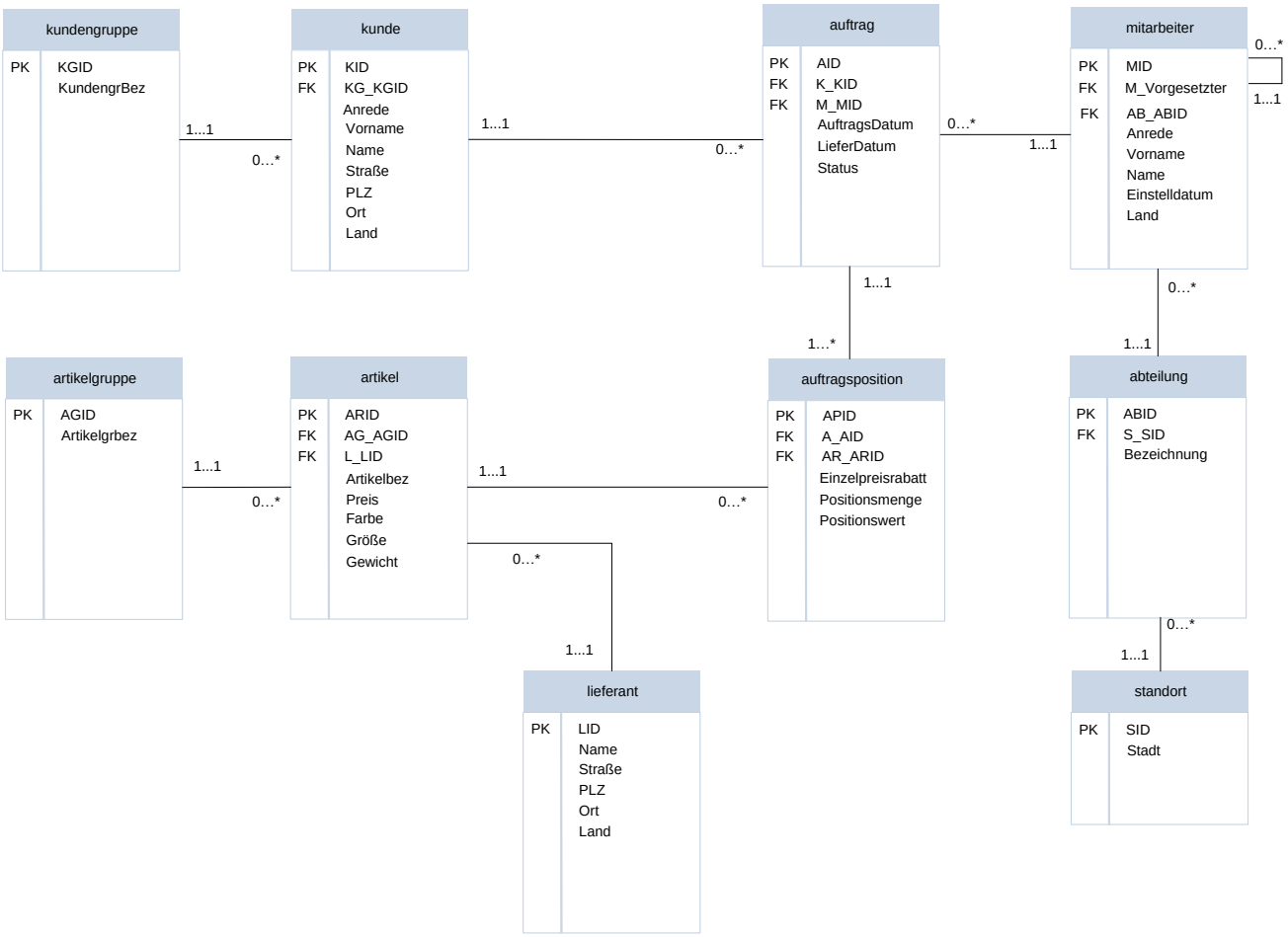


**Abbildung 2: Min-Max-Notation**

### Kommentar zur Kardinalität der Beziehungen:

Ein Kunde gehört zu genau einer Kundengruppe und eine **Kundengruppe** kann keinen oder mehrere Kunden enthalten.

Ein **Kunde** muss keinen Auftrag erteilen (Neukunde). Doch er kann, wenn er möchte, mehrere Aufträge erteilen.

Ein **Auftrag** dagegen kann nur zu genau einem Kunden bzw. zu genau einem Mitarbeiter gehören und enthält entweder eine oder mehrere Auftragspositionen.

Ein **Mitarbeiter** kann keine oder mehrere Aufträge erstellen.

Ein **Vorgesetzter** kann mehrere Mitarbeiter haben, doch es kann sein, dass ein Mitarbeiter keinen Vorgesetzten hat.

Jeder Mitarbeiter gehört zu genau einer Abteilung. Eine **Abteilung** kann keine Mitarbeiter haben, wenn die Abteilung beispielsweise neu gegründet wurde, oder sie kann mehrere Mitarbeiter haben. Eine Abteilung gehört zu genau einem Standort (Standort-ID) und ein **Standort** kann keine oder mehrere Abteilungen haben.

Eine **Auftragsposition** gehört zu genau einem Auftrag und enthält mindestens einen Artikel.

Ein **Artikel** kann in keinen oder mehreren Auftragspositionen vorkommen. Ein Artikel wird genau von einem Lieferanten geliefert und gehört zu genau einer Artikelgruppe.

Eine **Artikelgruppe** kann mehrere Artikel enthalten, doch wenn eine Artikelgruppe neu angelegt wird, enthält diese Artikelgruppe keine Artikel.

Ein **Lieferant** kann mehrere Artikel liefern, doch wenn ein Lieferant neu angelegt wird und noch keinen Auftrag zugeteilt bekommt, sind ihm noch keine Artikel zugeordnet.

## phpMyAdmin-Notation:

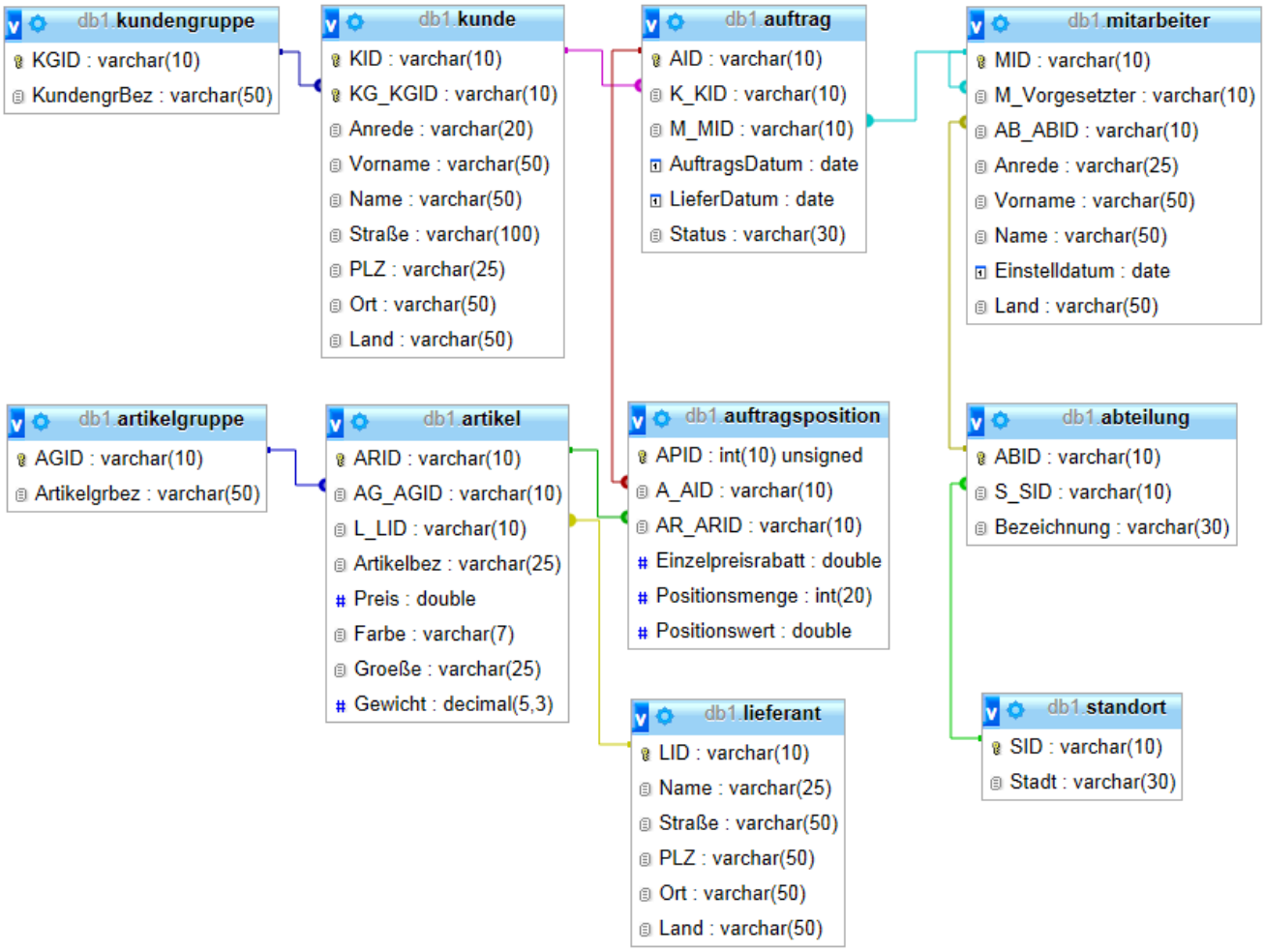


**Abbildung 3: phpMyAdmin Xampp-Notation**

Kommentar:

In phpMyAdmin gibt es bei MySQL keine festen Beziehungen wie bei Microsoft Access. Stattdessen wird erst mit dem JOIN die Beziehung in der jeweiligen Abfrage hergestellt oder im Tool phpMyAdmin wird eine Beziehungsübersicht[7] erstellt.

[7] Kapitel 2.2.1.4 phpMyAdmin - Primär- und Fremdschlüssel erstellen

## Krähenfußnotation:

Bei jeder Beziehung stehen zwei Kardinalitäten hintereinander, die das minimale bzw. das maximale Auftreten beschreiben:

| (Krähenfuß) | Min-Max | Bedeutung |
|---|---|---|
| ⟨ | 0 .. * bzw. 1…* | null oder viele bzw. eins oder viele |
| ‖ | 1 | eins |

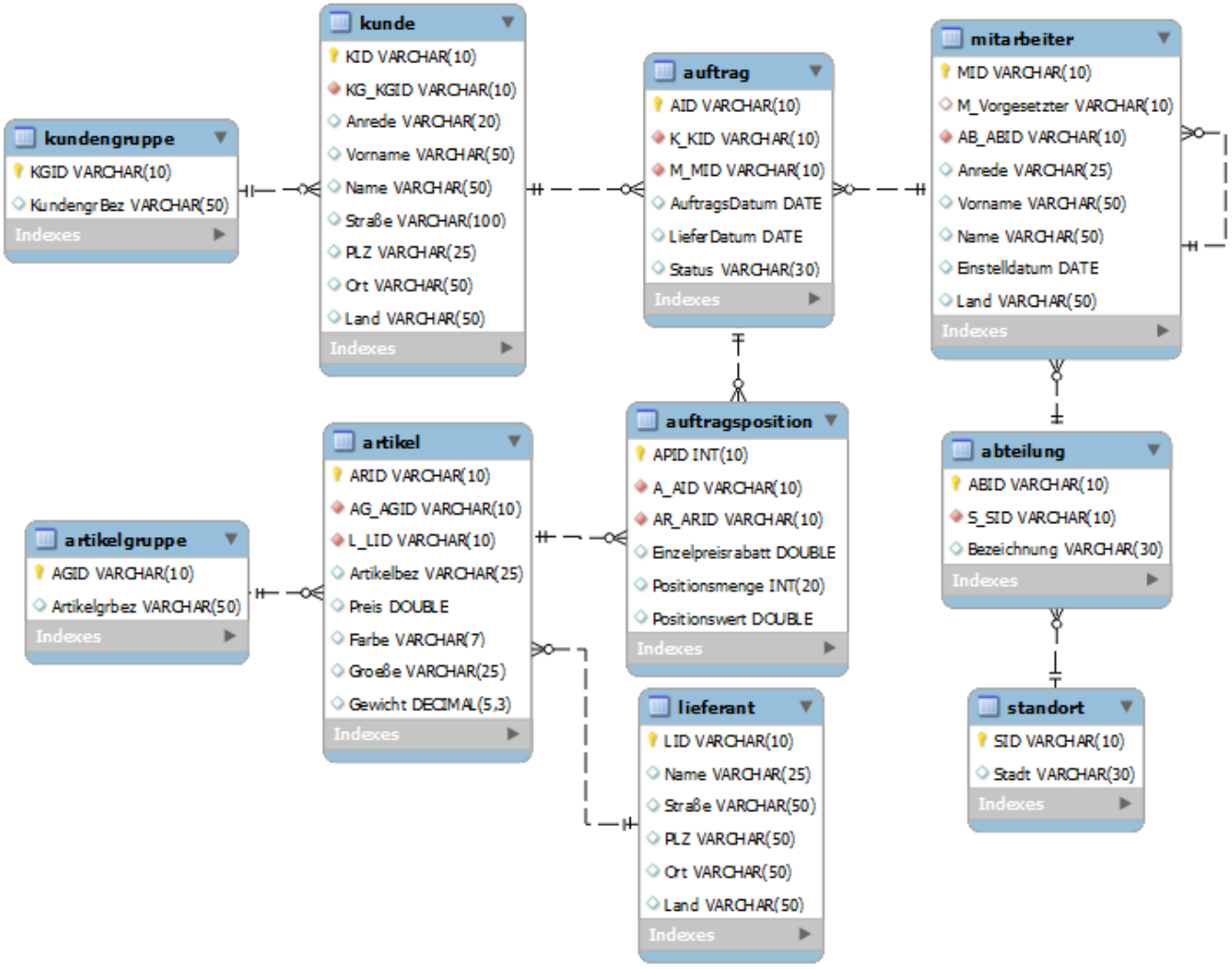


**Abbildung 4: MySQL Workbench Krähenfuß-Notation**

### Kommentar:

Die Darstellung wird mit dem Tool MySQL Workbench[8] erstellt. Diese Notation ist die selbe wie in der Min-Max-Notation, nur dass statt dem 0…* bzw. 1…* ein ⟨ -Zeichen und statt 1…1 ein ‖ - Zeichen zu sehen ist. Die genauere Vorgehensweise wird im Kapitel 2.2.2 MySQL Workbench beschrieben.

[8] Kapitel 2.2.2 MySQL Workbench

## 2.2 TOOLS

Um MySQL mit PHP auf dem Computer zu testen, wird ein eigener Server gebraucht. Der erste Schritt zum Lernen von MySQL ist es, auf dem Rechner einen Apache Server zu installieren. Damit kann eine Datenbank angelegt werden. Anschließend wird eine Verbindung mit der Datenbank aufgebaut und mit MySQL Workbench verbunden.

### 2.2.1 XAMPP

**XAMPP** ist eine Zusammenstellung von freier Software. XAMPP ermöglicht das einfache Installieren und Konfigurieren des Webservers Apache mit der Datenbank MySQL und den Skriptsprachen Perl und PHP. XAMPP ist ein Kunstwort und setzt sich neben dem „X“ aus den Anfangsbuchstaben der Technologien Apache, MySQL, PHP und Perl zusammen. Da es XAMPP nicht nur für ein Betriebssystem gibt, wurde das „X“ als Metapher für das Unbekannte definiert. Das "X" steht stellvertretend für das Betriebssystem, unter dem XAMPP läuft. Mit XAMPP wird die Möglichkeit geboten, auf ganz einfache Art und Weise Techniken wie Apache, MySQL, PHP und Perl ohne ein spezielles Vorwissen und ohne großen Aufwand auf den Rechner zu holen.

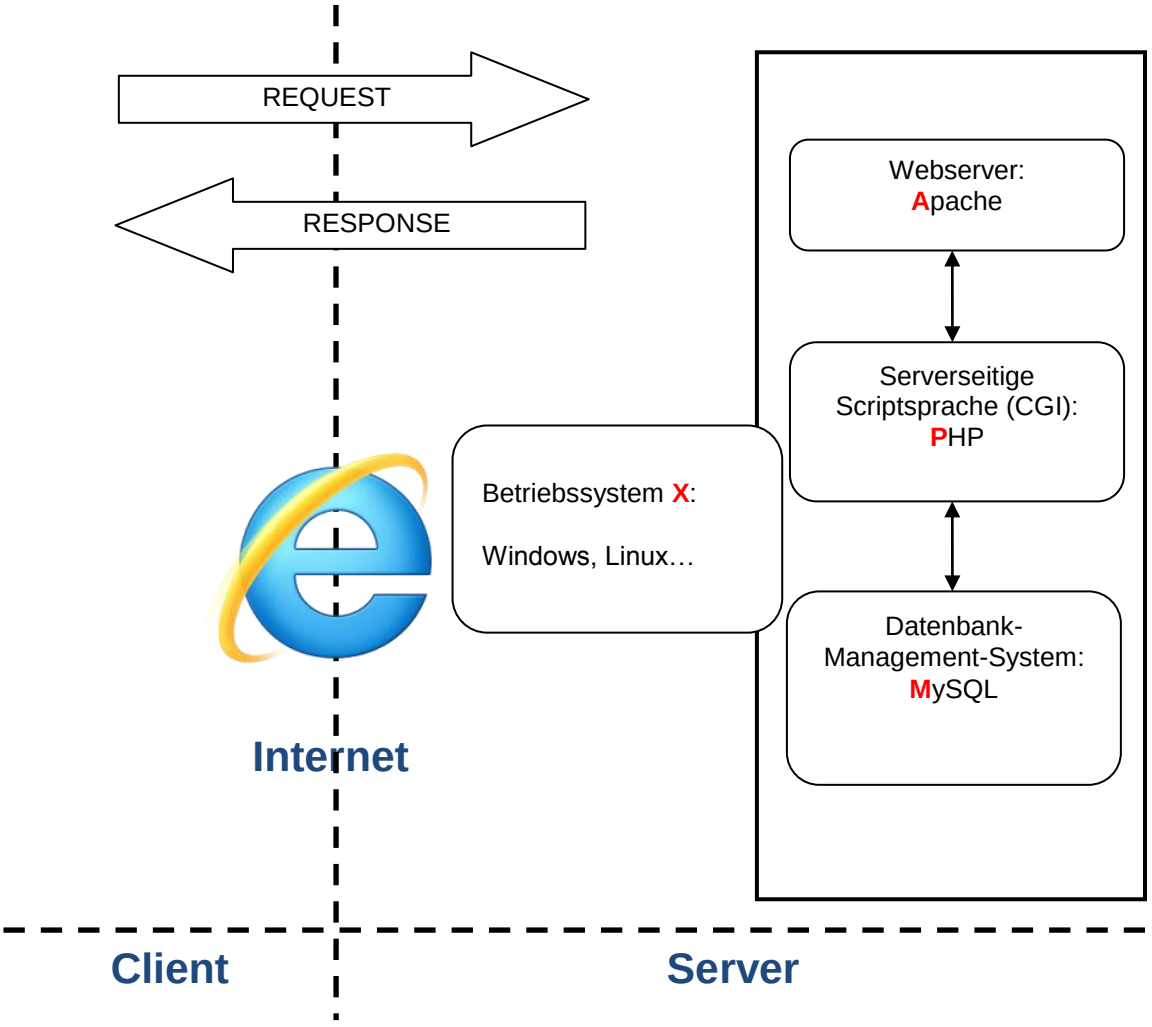


**Abbildung 5: XAMPP Architektur**

### 2.2.1.1 XAMPP INSTALLIEREN FÜR WINDOWS

Für die Installation sind die Administrationsrechte von Windows nötig!
Zurzeit gibt es vier XAMPP-Distributionen:

- XAMPP für Windows
- XAMPP für Linux
- XAMPP für Mac OS X
- XAMPP für Solaris

Sobald die Installation abgeschlossen ist, kann das XAMPP Control Panel geöffnet werden. Mittels diesem Konsolprogramm wird zunächst der Datenbankserver MySQL gestartet. Nach dem erfolgreichen Start von MySQL kann der eigentliche Web-Server Apache gestartet werden. Das folgende Bild zeigt sich nach erfolgreichem Start beider Applikationen.

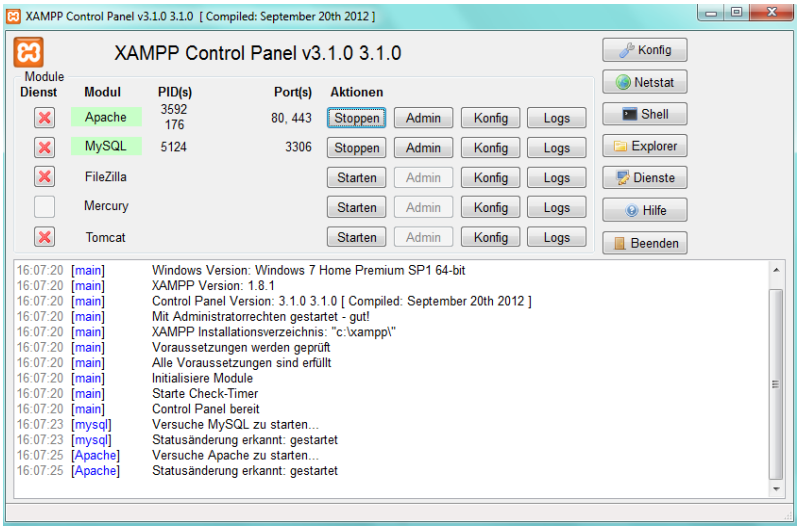


**Abbildung 6: XAMPP Control Panel**

Anschließend wird der Internet Explorer oder ein anderer Web-Browser gestartet und als URL „**http://localhost**“ eingegeben.

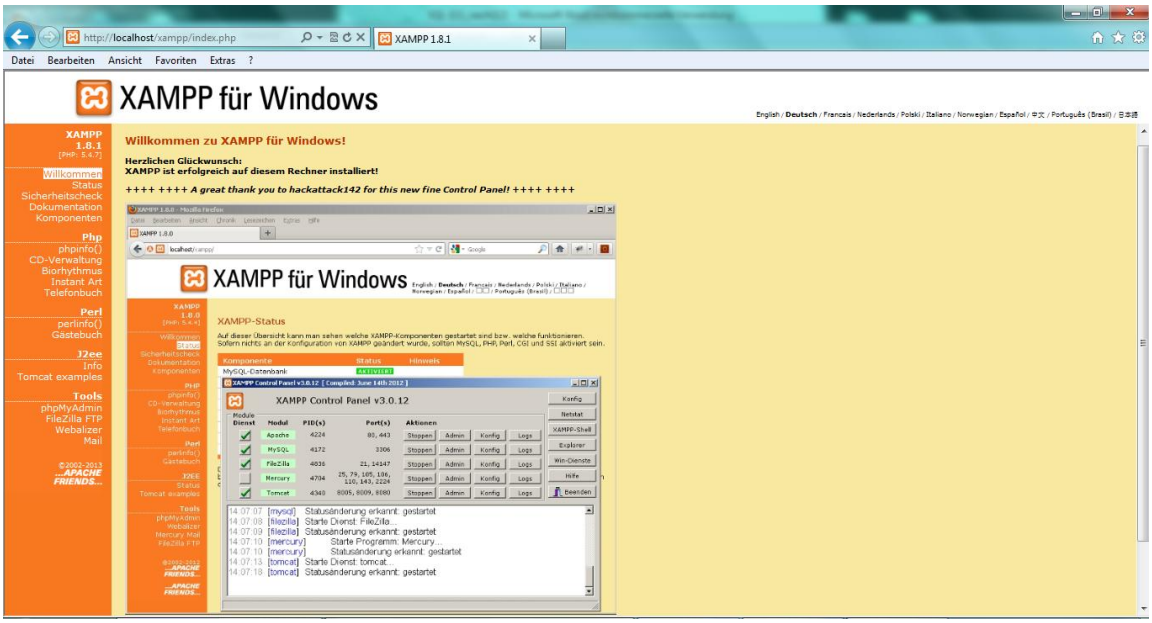


**Abbildung 7: http://localhost**

Es ist wichtig zu wissen, dass XAMPP nicht für den Produktionseinsatz, sondern nur für Entwickler in Entwicklungsumgebungen gedacht ist. Das hat zur Folge, dass XAMPP absichtlich nicht restriktiv sondern im Gegenteil sehr offen vorkonfiguriert ist. Für einen Entwickler ist das ideal, da er so keine Grenzen vom System vorgeschrieben bekommt. Für einen Produktionseinsatz ist das allerdings nicht geeignet. Dinge, die an XAMPP absichtlich(!) unsicher sind:

- Der MySQL-Administrator (root) hat kein Passwort
- Der MySQL-Daemon ist über das Netzwerk erreichbar
- phpMyAdmin ist über das Netzwerk erreichbar
- XAMPP Verzeichnis ist nicht geschützt

Alle aufgeführten Punkte können zu schwerwiegenden Sicherheitsproblemen führen, wenn der betreffende Rechner schutzlos und damit für jede außen stehende Person zugänglich im Internet agiert. Diese Lücken können bei Bedarf geschlossen werden. In vielen Fällen reicht hierzu eine Firewall oder einfach eine Internetverbindung über einen externen Router aus. In beiden Fällen ist der Rechner in der Regel nicht von außen erreichbar. Eine erste Hilfe bietet die "XAMPP Sicherheitskonsole".

### 2.2.1.2 PHPMYADMIN – DATENBANK ANLEGEN

Im linken Menü unter **Tools** befindet sich der Link **phpMyAdmin**. In diesem Tool kann eine Datenbank angelegt werden, dafür wird auf der Sekundär-Navigation (Abbildung 9) auf „Datenbanken“ geklickt. Darunter befindet sich ein Eingabefeld, in welches der Name der Datenbank geschrieben wird.

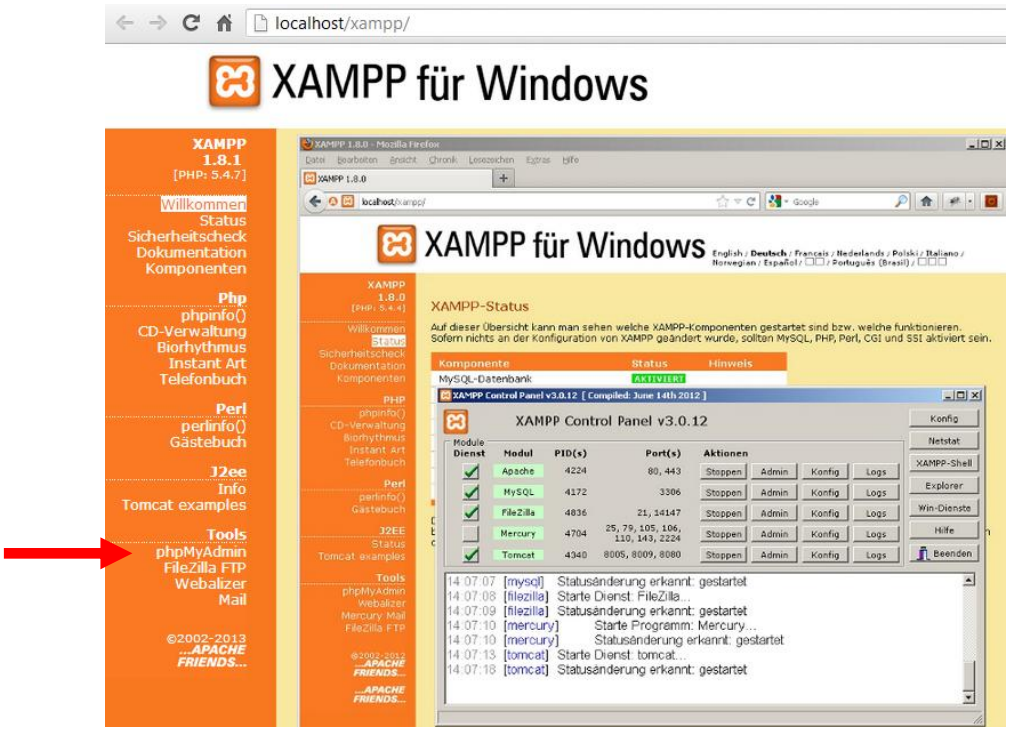


**Abbildung 8: XAMPP phpMyAdmin starten**

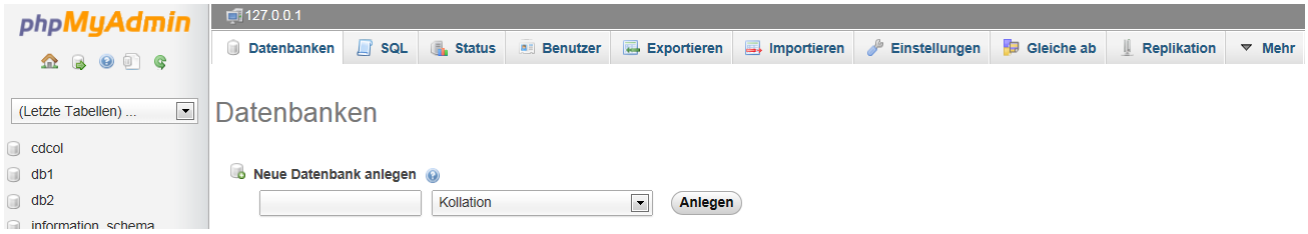


**Abbildung 9: Datenbank erstellen**

### 2.2.1.3 PHPMYADMIN – TABELLEN ERZEUGEN

In diesem Teil werden in der Datenbank Tabellen erstellt, Werte eingetragen und SQL-Abfragen durchgeführt. Anschließend wird mit PHP eine Verbindung zur Datenbank hergestellt[9]. Unter dem Text „Erzeuge Tabelle“ befindet sich ein Eingabefeld, in welches der Name der Tabelle eingetragen wird.

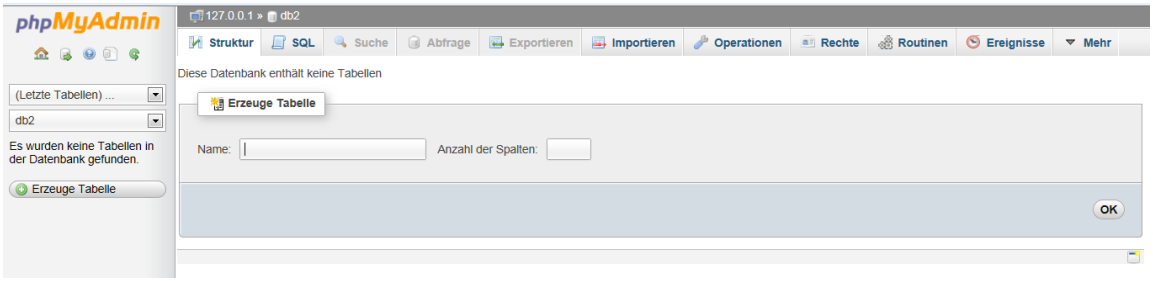


**Abbildung 10: Tabellen erzeugen**

In der Datenbank wird die erste Tabelle „kunde“ erstellt. Die Anzahl der Spalten wird rechts neben dem Namen eingegeben. Jede Tabelle kann von mindestens eine bis zu beliebig vielen Spalten haben. Hier wird der Wert 9 eingegeben. Die neun Felder müssen nun im folgenden Formular korrekt gefüllt werden, erst danach wird die Tabelle erzeugt. Die erste Spalte ist eine alphanumerische „KID“ mit der Länge 10; der Wert der Spalte, also der Primärschlüssel, darf nicht Null (leer) sein. Die anderen acht Spalten „KG_KGID“, „Anrede“, „Vorname“, „Name“, „Straße“, „PLZ“, „Ort“ und „Land“ dagegen dürfen Null sein und haben eine beliebige alphanumerische Länge. Alphanumerisch deswegen, weil die Schlüssel und die Attribute nicht nur Zahlen sondern auch Sonderzeichen beinhalten können. Anschließend wird gespeichert.

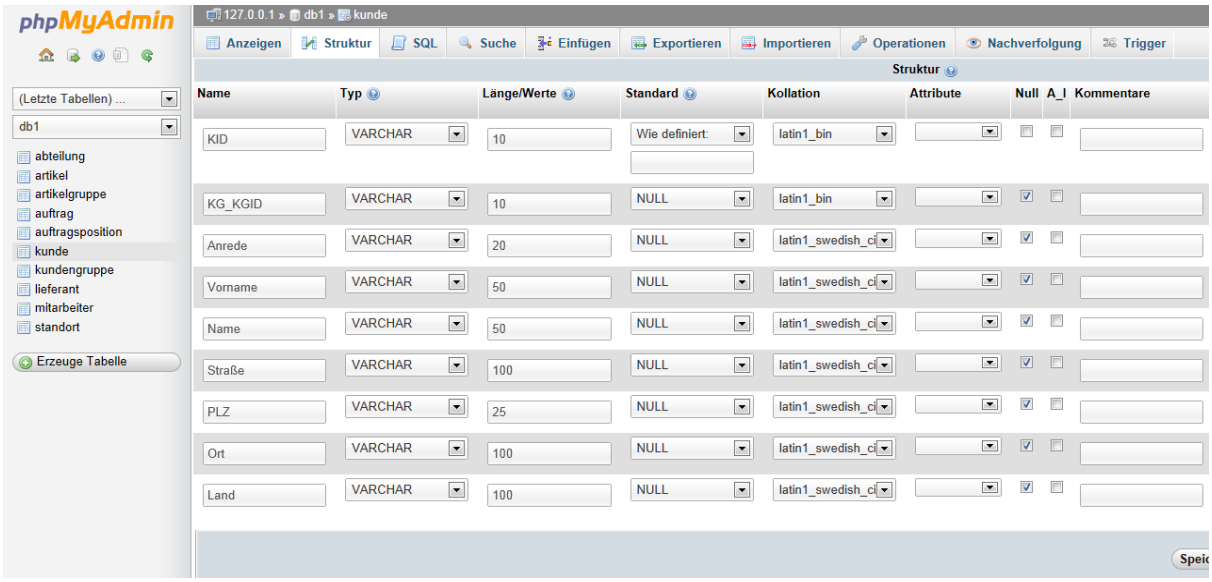


**Abbildung 11: erste Tabelle "Kunde"**

[9] 2.2.1.6 phpMyAdmin - Verbindung zur Datenbank herstellen

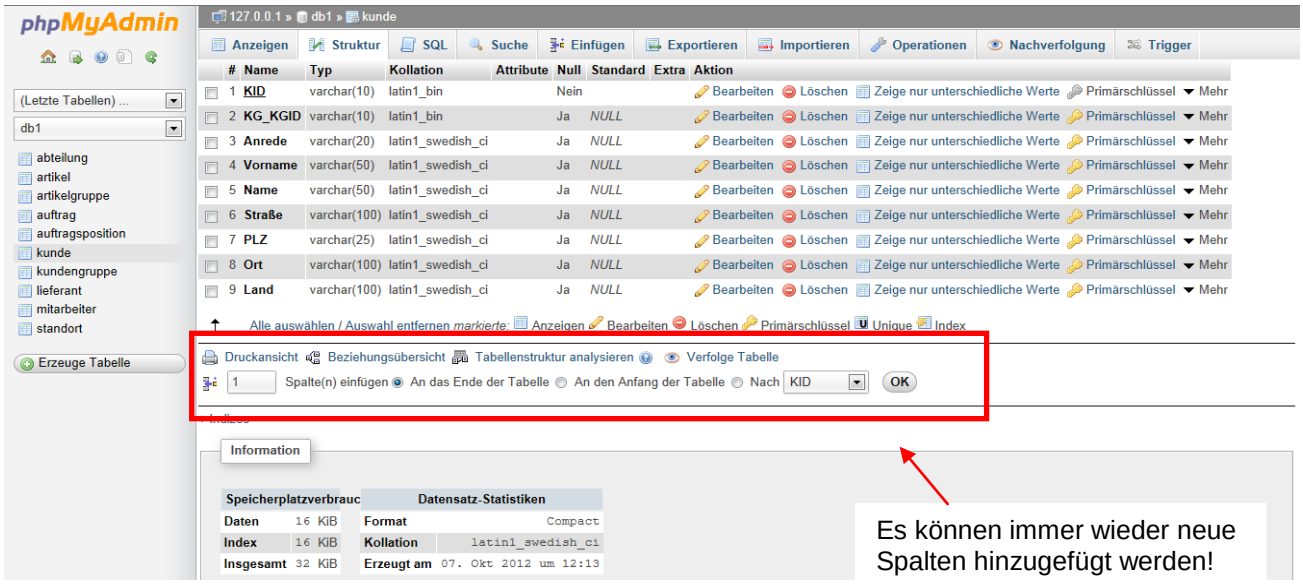


**Abbildung 12: Weitere Spalten hinzufügen**

Die erste Tabelle ist angelegt, und es können weitere Tabellen erstellt werden.

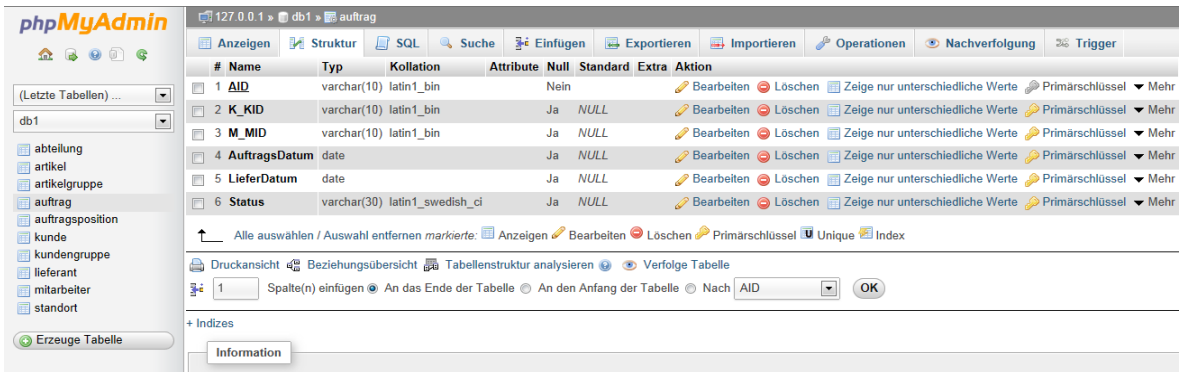


**Abbildung 13: Struktur der Tabelle "auftrag"**

### 2.2.1.4 phpMyAdmin – Primär- und Fremdschlüssel erstellen

Für das Erstellen der Schlüssel wird eine weitere Tabelle „auftrag“ erzeugt.
Um den Primärschlüssel zu erstellen, wird rechts im Bild (Abbildung 13) auf den gelben Schlüssel in der Spalte „KID“ geklickt und somit ist der Primärschlüssel definiert. Genauso wird in der Tabelle „auftrag“ in der Spalte „AID“ der Primärschlüssel definiert.
Um den Fremdschlüssel zu erstellen, wird die Tabelle angeklickt, in der der Fremdschlüssel liegen soll. In diesem Beispiel wird auf die Tabelle „auftrag“ geklickt, ein Haken im rechten Kasten neben der Spalte „K_KID“ gemacht und anschließend auf „**Beziehungsübersicht**“ geklickt.

### 2.2.1.5 phpMyAdmin – Werte in die erzeugten Tabellen eintragen

Nachdem alle Tabellen erstellt und die Schlüssel vergeben sind, können die Formulare ausgefüllt werden.

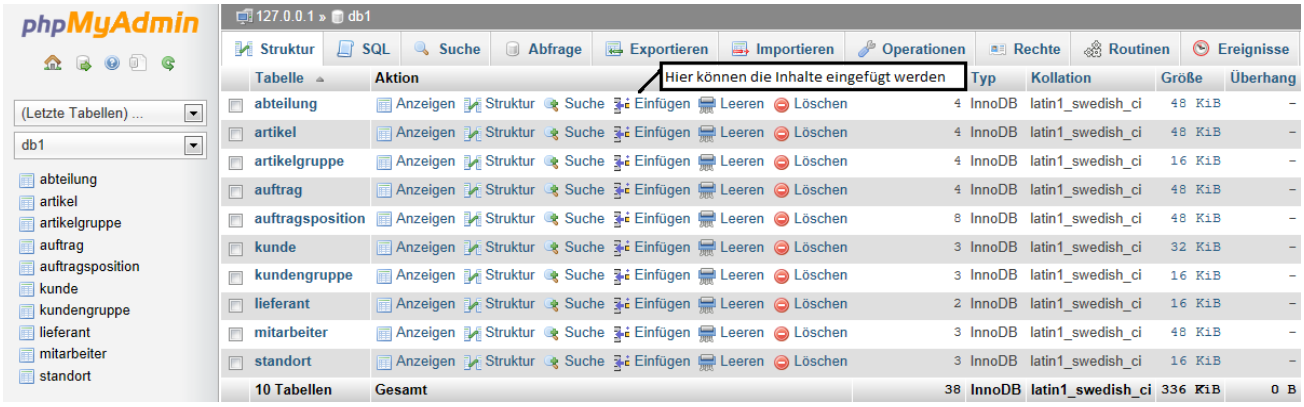


**Abbildung 14: phpMyAdmin Tabellenwerte einfügen**

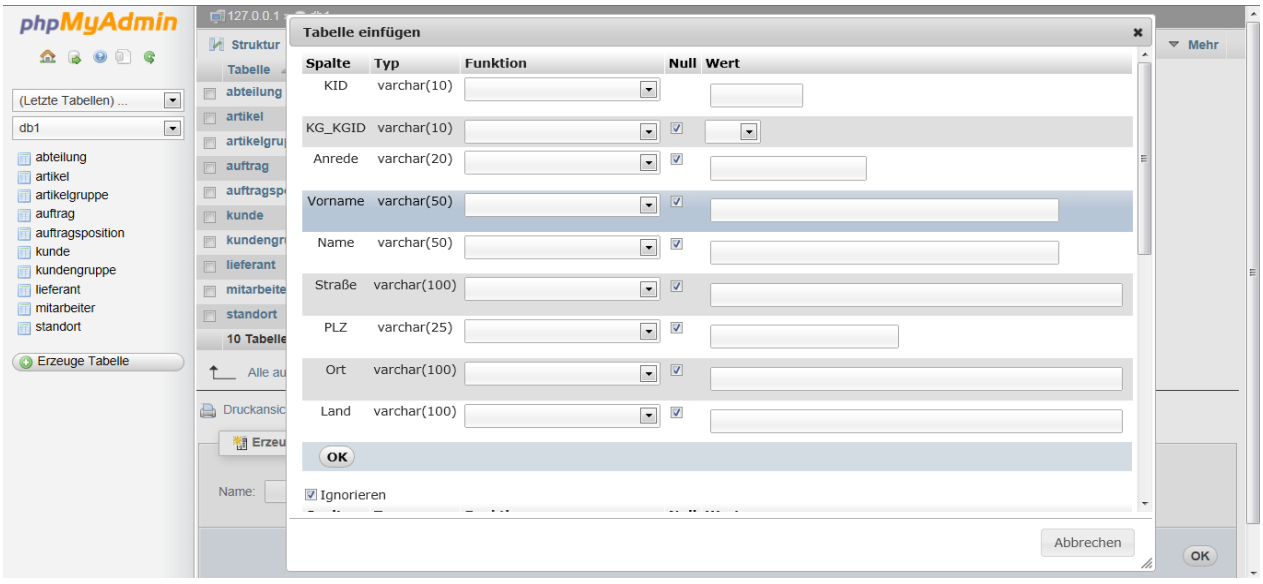


**Abbildung 15: Schlüssel und Attribute ausfüllen**

Die meisten Spalten sind vom **Typ varchar** (alphanumerisch), das heißt die Werte erwarten Strings als Input. Die Spalte Funktion kann ignoriert werden. Bei varchar steht in Klammern eine Zahl, sie gibt an, wie lang der jeweilige String jeweils sein darf. Der Primärschlüssel KID darf nicht länger als 10 Zeichen sein, der Name des Kunden darf nicht länger als 50 Zeichen sein usw. Die restlichen Spalten der Tabellen haben die Datentypen **int**, **double**, **date** und **decimal** (für 3 Nachkommastellen). Nun existiert eine Datenbank, und SQL-Befehle können ausgeführt werden.
Unter dem Punkt „Mehr“ und „Designer“ kann die XAMPP-Notation[10] dargestellt werden.

**Abbildung 16: phpMyAdmin Tabellenübersicht**

### 2.2.1.6 phpMyAdmin – Verbindung zur Datenbank herstellen

Um SQL Befehle auszuführen, wird mit PHP auf die Datenbank zugegriffen. MySQL wurde bereits im XAMPP gestartet. Um die Verbindung aufzubauen, wird auf den Button „Admin“[11] geklickt. Es wird nach einem Benutzernamen und einem Passwort gefragt. Der Benutzername ist "root" und das Passwortfeld muss leer bleiben.

Alternativ kann auch mit dem folgenden PHP-Code eine Verbindung zur Datenbank hergestellt werden:

```
<?php
    mysql_connect("localhost", "Benutzername","Passwort");
    mysql_select_db("Datenbank-Name");
?>
```

Der PHP-Befehl “mysql_connect” öffnet eine Verbindung zu einem MySQL-Server. “localhost” ist in der Regel der richtige Wert, außer das PHP-Skript liegt auf einer anderen Domain, dann müsste von “domain-a.de” (auf der das Skript liegt) auf eine Datenbank von “domain-z.de” zugegriffen werden, und anstatt “localhost” müsste der Wert “domain-z.de” eingetragen werden. Benutzername und

[10] Kapitel 2.1 Datenmodell in 4 Darstellungen
[11] Kapitel 2.2.1.2 phpMyAdmin – Datenbank anlegen

Passwort sind die Datenbank-Logindaten, die werden meistens automatisch generiert, wenn die Datenbank angelegt wird. Wenn PHPMyAdmin auf dem eigenen Computer installiert wurde, ist der default-Wert für den Benutzernamen “root”, und das Passwort ist nicht gesetzt, also:

```
<?php
    mysql_connect("localhost", "root","") or die ("Verbindung nicht
möglich");
?>
```

Mit dem PHP-Befehl “mysql_select_db” wird die Datenbank ausgewählt, auf die zugegriffen werden soll. Ein MySQL-Server kann mehr als eine Datenbank haben, deshalb muss das mit angeben werden. Mit diesen beiden Befehlen würde eine Verbindung zur Datenbank stehen. Beim Einrichten können Fehler gemacht werden. Dafür sollte eine Fehlermeldung ausgegeben werden:

```
<?php
    mysql_connect("localhost", "Benutzername", "Passwort") or die
("Verbindung nicht
       möglich");
    mysql_select_db("Datenbank-Name") or die ("Datenbank existiert nicht");
?>
```

Dadurch kann der Fehler gefunden werden, falls ein Wert falsch eingetragen wurde oder die Datenbank nicht existiert.

### 2.2.2 MySQL Workbench

Neben dem Ausführen von SQL-Abfragen in phpMyAdmin besteht die Möglichkeit, ein Desktop-Programm, welches direkt auf die Datenbank zugreift, zu installieren, um damit SQL-Befehle auszuführen. Dies ist nicht erforderlich, kann aber zusätzlich erstellt werden, um zu sehen, wie eine externe Verbindung aufgebaut wird. Diese Verbindung übernimmt die Tabellen, und die Abfrage kann durchgeführt werden.

Die **MySQL Workbench** ermöglicht, beliebige Arten von Datenbanken zu entwerfen, zu modellieren, zu erstellen und zu verwalten. Die MySQL Workbench enthält alle Funktionen, die ein Datenmodellierer für den Entwurf komplexer ER-Modelle sowie für ein Forward- und Reverse-Engineering benötigt. Darüber hinaus stellt die MySQL Workbench wichtige Funktionen zur Durchführung schwieriger Änderungsmanagement- und Dokumentationsaufgaben bereit, die normalerweise viel Zeit und Mühe erfordern. Doch in diesem Kapitel sollen SQL-Befehle durchgeführt werden, dafür muss nur eine Verbindung zwischen XAMPP und MySQL Workbench existieren. Auf die restlichen Funktionen der MySQL Workbench wird nicht näher eingegangen.

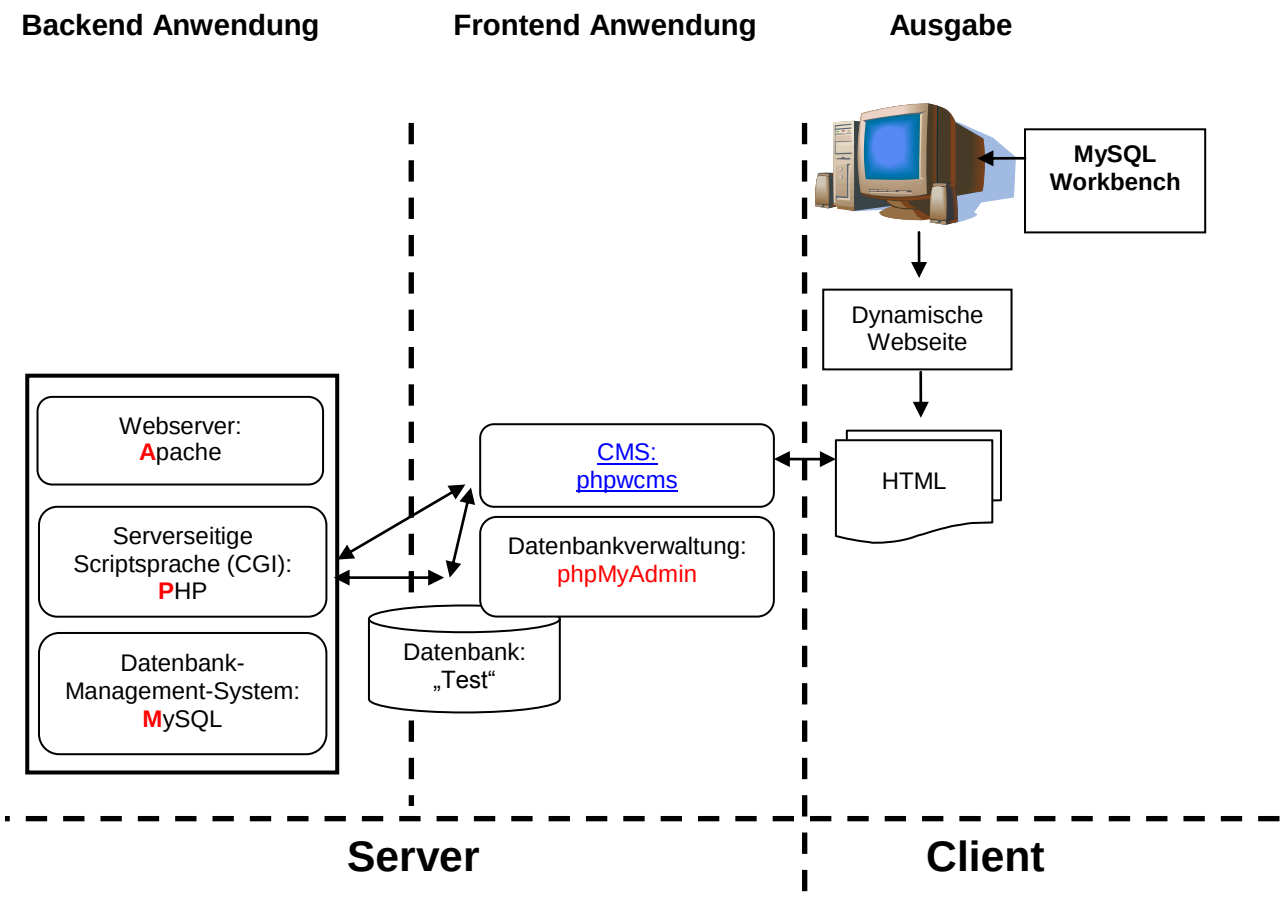


**Abbildung 17: XAMPP und MySQL Workbench Architektur**

### 2.2.2.1 MySQL Workbench installieren

Zunächst wird MySQL Workbench heruntergeladen und ausgeführt. Anschließend kann MySQL Workbench geöffnet werden. Auf der Startseite wird auf den Button „New Connection“ angeklickt. Der Connection Name kann willkürlich genannt werden, hier „DB1“. Hostname ist die IP-Adresse, und Username ist, wie oben bereits erwähnt, „root“. Die Verbindung wird mit „OK“ fortgesetzt.

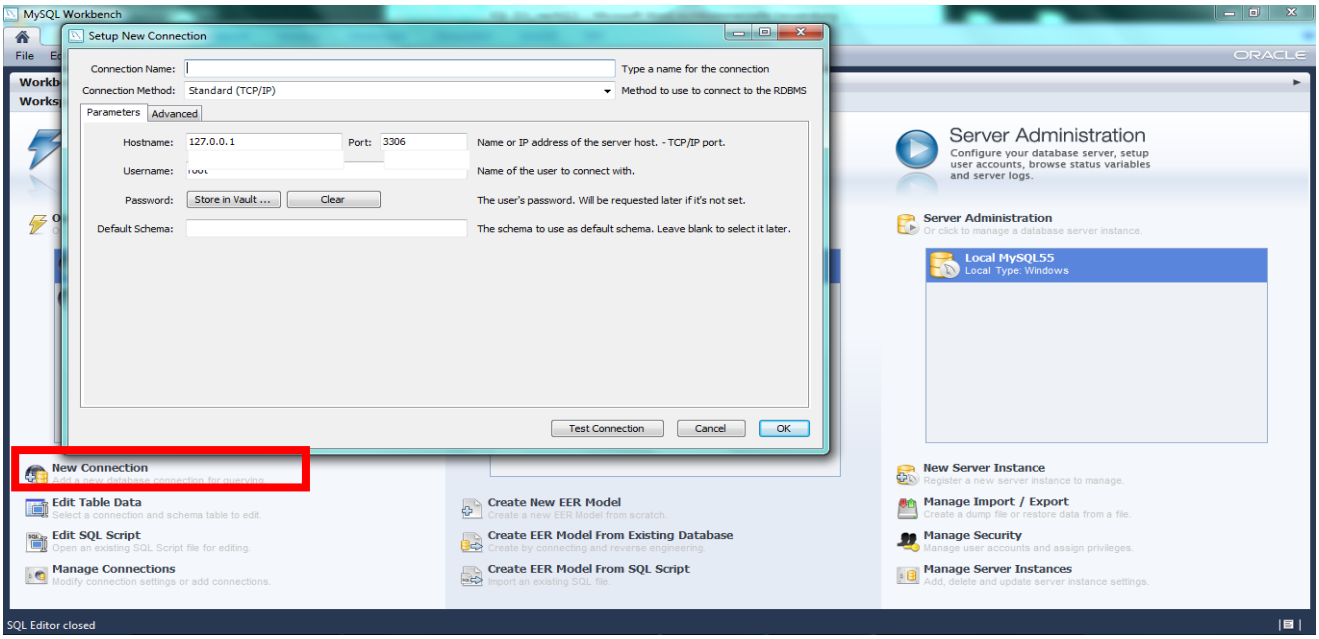


**Abbildung 18: MySQL Workbench Connection**

Mit einem Doppelklick auf die Verbindung „DB1“ wird MySQL Workbench nun mit der Datenbank geöffnet. Nach der Verbindung entsteht dieses Bild in MySQL Workbench:

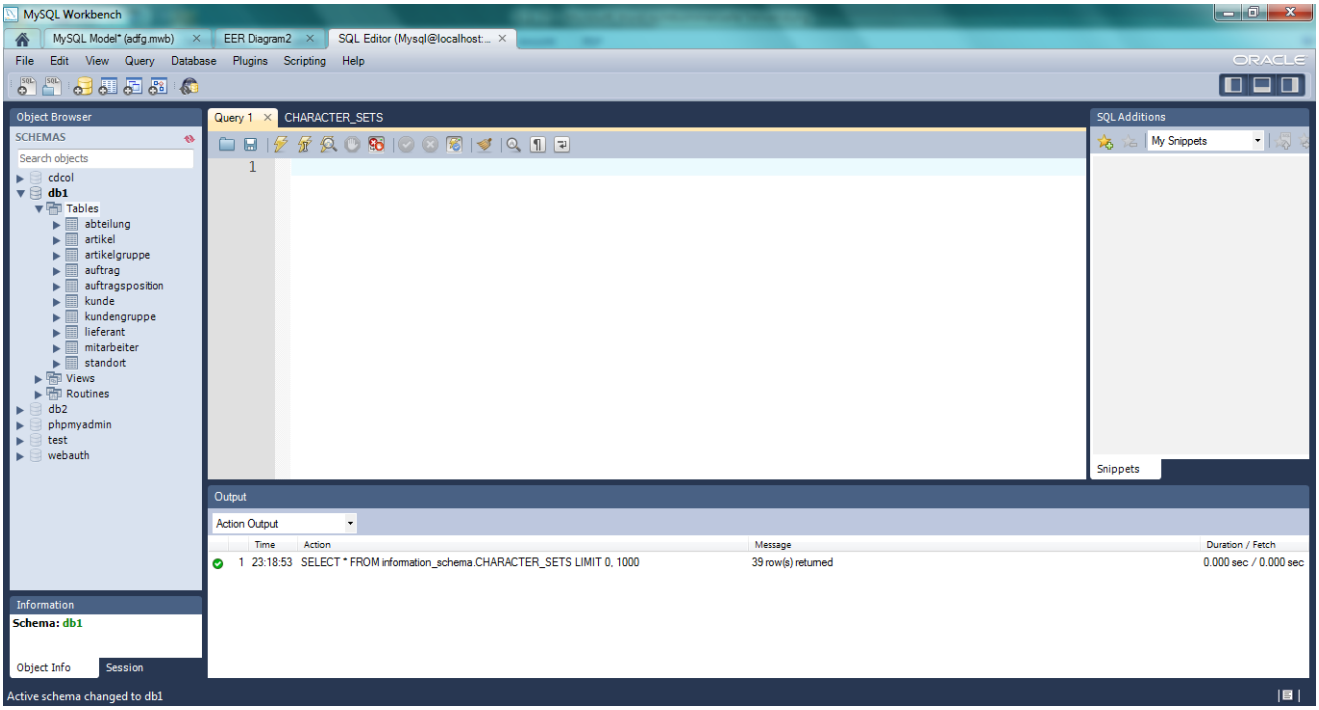


**Abbildung 19: MySQL Workbench Abfragefenster**

Die Tabellen aus dem Tool phpMyAdmin wurden somit übertragen, und die SQL-Abfrage kann eingegeben werden.

### 2.2.2.2 MySQL Workbench – Krähenfußnotation

Die Tabellen können als Krähenfuß-Notation dargestellt werden, hierzu wird auf den Punkt „Add Diagram“ gedrückt. Darin können alle Tabellen auf die Fläche gezogen werden. Somit entsteht die Krähenfuß-Notation in MySQL Workbench.

Die zweite Möglichkeit, die Krähenfuß-Notation anzeigen zu lassen, ist, die Verbindung mit dem EER-Model herzustellen. Der Vorteil dieser Variante ist, dass nicht nur die Krähenfuß-Notation angezeigt wird, sondern dass die Tabellen gleich miterstellt werden. Dazu wird MySQL Workbench gestartet und eine Verbindung zur Datenbank hergestellt:

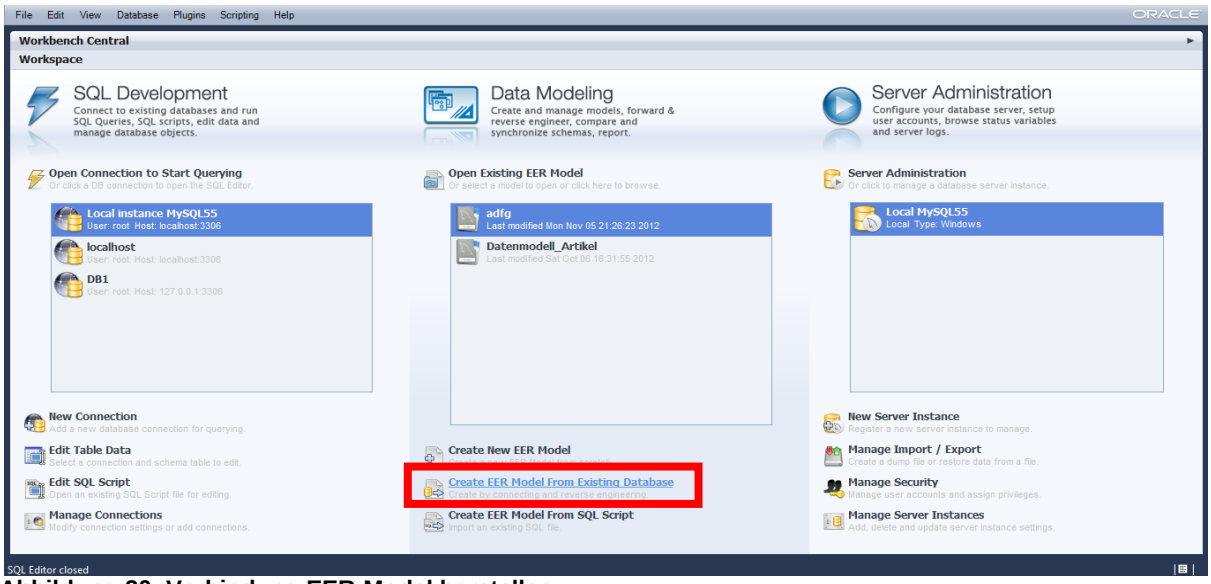


**Abbildung 20: Verbindung EER-Model herstellen**

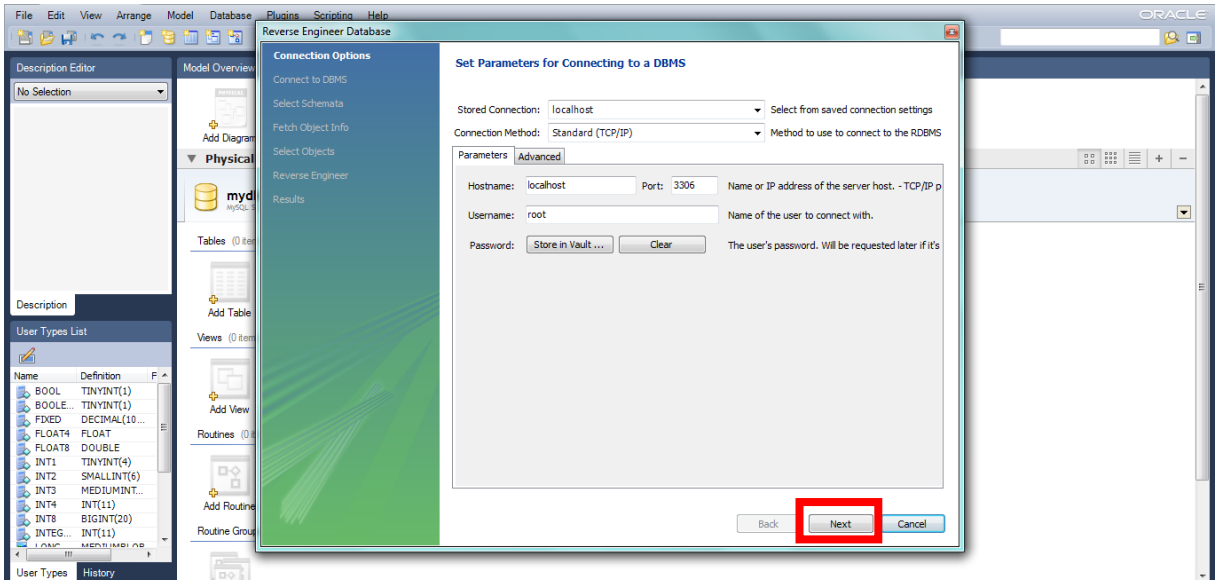


**Abbildung 21: Connecting-Daten ausfüllen**

Es wird so lange auf Next gedrückt, bis dieses Bild zu sehen ist:

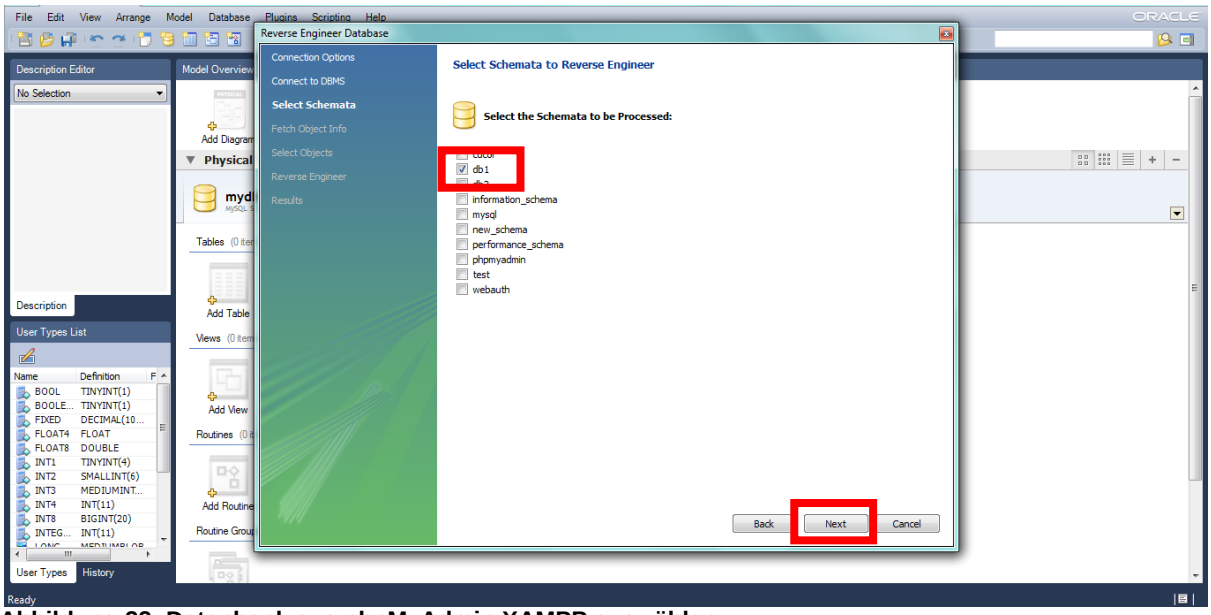


**Abbildung 22: Datenbank aus phpMyAdmin XAMPP auswählen**

### 2.2.2.3 MySQL Workbench – Primär- und Fremdschlüssel erstellen

Da es in phpMyAdmin keine Beziehungen gibt, können sie nicht in MySQL Workbench übertragen werden. In phpMyAdmin können lediglich Primär- und Fremdschlüssel vergeben werden. Die Beziehungen werden in MySQL Workbench im Nachhinein richtig gestellt. Das heißt, sobald die Verbindung aufgebaut ist, öffnen sich automatisch die Tabellen und die dazugehörige Krähenfuß-Notation. Die Voreinstellung der Tabellen ist standardmäßig eingestellt. Durch einen Doppelklick auf die Beziehung öffnet sich die Eigenschaft der Beziehung:

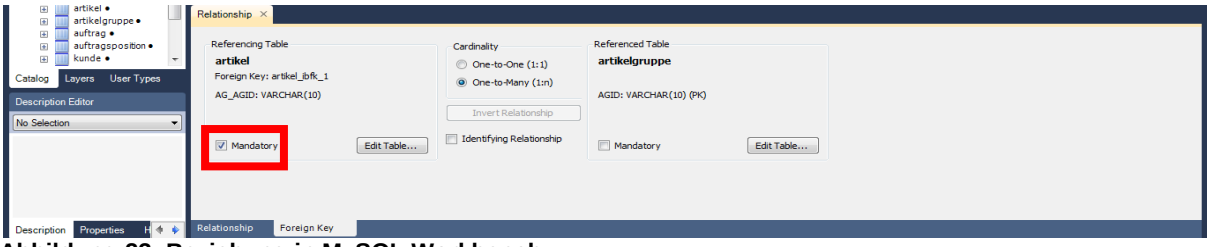


**Abbildung 23: Beziehung in MySQL Workbench**

Der Haken ist immer in der linken Tabelle eingestellt, das heißt die linke Tabelle ist immer auf 1…1 und die rechte 1…* gesetzt. Die Beziehung zwischen den zwei Tabellen und die interne Beziehung muss manuell geändert werden. Sobald die Änderungen der einzelnen Beziehungen durchgeführt sind, wird die Krähenfuß-Notation richtig dargestellt.

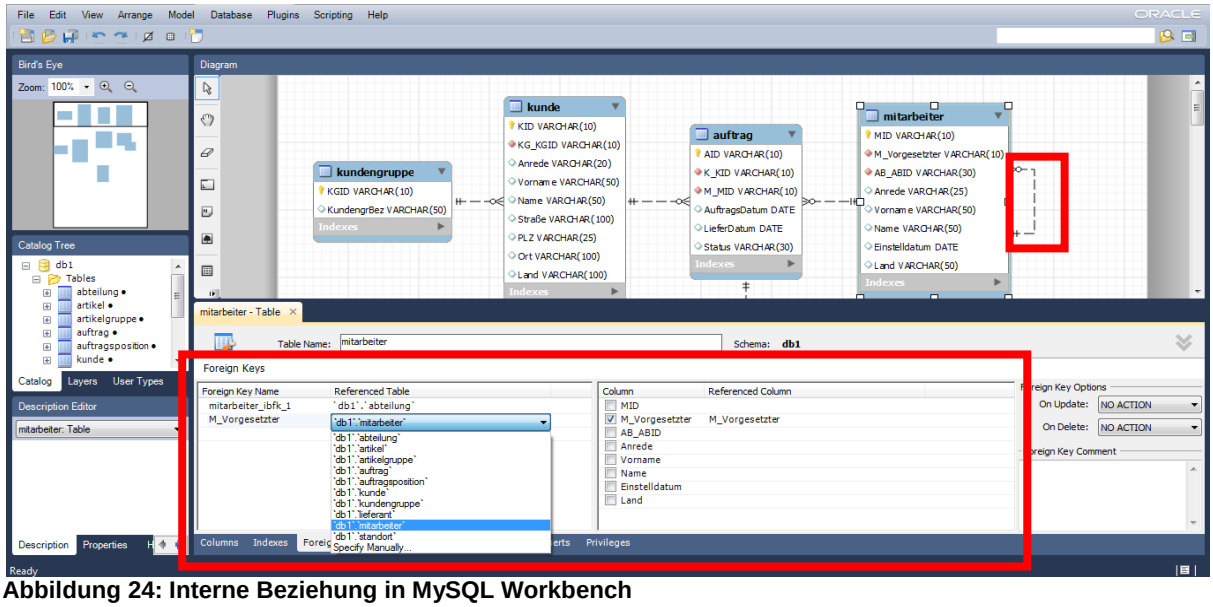


**Abbildung 24: Interne Beziehung in MySQL Workbench**

Für das Einfügen der internen Beziehung wird mit einem Doppelklick die Tabelle geöffnet. Links wird der Fremdschlüsselname gewählt und rechts die zugehörige Tabelle. Erst dann ist die interne Beziehung definiert. *(Krähenfuß-Notation Kapitel 2.1)*

## 2.3 TESTDATEN

**abteilung**

| ABID | S_SID | Bezeichnung |
|---|---|---|
| 1.1 | S1 [->] | Verkauf |
| 1.2 | S2 [->] | Verwaltung |
| 1.3 | S1 [->] | Wartung |
| 1.4 | S3 [->] | Lager |
| 1.5 | *NULL* | Kundendienst |

**artikel**

| ARID | AG_AGID | L_LID | Artikelbez | Preis | Farbe | Groeße | Gewicht |
|---|---|---|---|---|---|---|---|
| 1a | A [->] | L1 [->] | ProBook | 400 | schwarz | 13,3 Zoll | 2.040 |
| 2a | D [->] | L1 [->] | Enterprise_500 | 200 | weiß | 470x440x160 mm | 42.200 |
| 3a | B [->] | L2 [->] | Scanjet | 150 | schwarz | 60x280x460 mm | 2.200 |
| 4a | C [->] | L2 [->] | Touch | 100 | schwarz | 50x111x36 mm | 0.088 |
| 5a | D [->] | L1 [->] | Officejet | 150 | grau | 350x448x206 mm | *NULL* |
| 6a | *NULL* | L2 [->] | SpaceMouse | 200 | *NULL* | *NULL* | *NULL* |

**artikelgruppe**

| AGID | Artikelgrbez |
|---|---|
| A | Laptop |
| B | Scanner |
| C | Maus |
| D | Drucker |
| E | Tastatur |

**auftrag**

| AID | K_KID | M_MID | AuftragsDatum | LieferDatum | Status |
|---|---|---|---|---|---|
| 1A | k1 [->] | *NULL* | 2013-07-11 | 2013-08-22 | offen |
| 2A | k2 [->] | M2 [->] | 2012-10-05 | 2012-11-30 | geliefert |
| 3A | k2 [->] | M1 [->] | 2013-05-22 | 2013-06-11 | *NULL* |
| 4A | k3 [->] | M1 [->] | 2012-10-09 | 2012-10-31 | geliefert |

**auftragsposition**

| APID | A_AID | AR_ARID | Einzelpreisrabatt | Positionsmenge | Positionswert |
|---|---|---|---|---|---|
| 1 | 1A [->] | 2a [->] | 10 | 3 | 540 |
| 2 | 1A [->] | 4a [->] | *NULL* | 5 | 500 |
| 3 | 2A [->] | 1a [->] | 5 | 1 | 380 |
| 4 | 2A [->] | 3a [->] | *NULL* | 3 | 450 |
| 5 | 3A [->] | 1a [->] | 5 | 1 | 380 |
| 6 | 3A [->] | 2a [->] | 10 | 2 | 360 |
| 7 | 3A [->] | 3a [->] | *NULL* | 1 | 150 |
| 8 | 4A [->] | 4a [->] | *NULL* | 4 | 400 |

**kunde**

| KID | KG_KGID | Anrede | Vorname | Name | Straße | PLZ | Ort | Land |
|---|---|---|---|---|---|---|---|---|
| k1 | GR2 [->] | Frau | Susanne | Herrmann | Lange Zeile 88 | 91052 | Erlangen | Deutschland |
| k2 | *NULL* | Herr | Gerd | Schneider | Triebstraße 11a | 80993 | München | Deutschland |
| k3 | GR3 [->] | Herr | Gerhard | Kofler | Schönbrunner Straße 94 | 1050 | Wien | Österreich |

**kundengruppe**

| KGID | KundengrBez |
|---|---|
| GR1 | A |
| GR2 | B |
| GR3 | C |
| GR4 | D |

**lieferant**

| LID | Name | Straße | PLZ | Ort | Land |
|---|---|---|---|---|---|
| L1 | HP | Herrenberger Straße 140 | 71034 | Böblingen | Deutschland |
| L2 | Microsoft | Box 27 | S-164 93 | Kista | Schweden |

**mitarbeiter**

| MID | M_Vorgesetzter | AB_ABID | Anrede | Vorname | Name | Einstelldatum | Land |
|---|---|---|---|---|---|---|---|
| M1 | *NULL* | 1.1 [->] | Herr | Norbert | Bauer | 2008-03-12 | Deutschland |
| M2 | M1 [->] | 1.2 [->] | Frau | Carola | Graf | 2009-07-22 | Deutschland |
| M3 | M4 [->] | 1.3 [->] | Herr | Hubert | Schulze | 2010-01-12 | Deutschland |
| M4 | *NULL* | 1.3 [->] | Frau | Simone | Stein | 2007-04-07 | Deutschland |

**standort**

| SID | Stadt |
|---|---|
| S1 | Nürnberg |
| S2 | Köln |
| S3 | Berlin |

**Abbildung 25: Testdaten**

| | Kapitel 3.1 | Kapitel 3.2 | Kapitel 3.3 | Kapitel 3.4 |
|---|---|---|---|---|
| **Woher?**<br>**Ausgangspunkt**<br>**Problemstellung** | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des istgleich-Operators (**=**) erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des ungleich-Operators (!=) erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des kleiner-Operators (**<**) erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des größer-Operators (**>**) erlernen und üben können. |
| **Wohin?**<br>**Zieldefinition** | Es soll eine Anleitung erstellt werden, die die Problemstellung löst. | analog | analog | analog |
| **Wie?**<br>**Methoden**<br>**(-verwendung)** | Der istgleich-Operator (=) ist theoretisch und anhand von Beispielen zu erklären. Übungsaufgaben für eine bestimmte Anzahl wesentlicher Befehlsvarianten sind zu entwerfen. | analog | analog | analog |
| **Was?**<br>**Resultate** | Diese Anleitung enthält die Befehlsvariante =. Die Variante wird theoretisch erklärt, und dazu wird ein Beispiel mit Kommentierung angegeben. Für die Studierenden sind zwei Übungsbeispiele angegeben, die sie selbstständig lösen und kommentieren sollen. | analog | analog | analog |

| | Kapitel 3.5 | Kapitel 3.6 | Kapitel 3.7 | Kapitel 3.8 | Kapitel 3.9 | Kapitel 3.10 |
|---|---|---|---|---|---|---|
| **Woher?**<br>**Ausgangspunkt**<br>**Problemstellung** | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des kleinergleich-Operators (**<=**) erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des größergleich-Operators (**>=**) erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **BETWEEN-AND**-Operators erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **LIKE**-Operators erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des IS **NULL**-Operators erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **IN**- und **NOT IN**-Operators erlernen und üben können. |
| **Wohin?**<br>**Zieldefinition** | analog | analog | analog | analog | analog | analog |
| **Wie?**<br>**Methoden (-verwendung)** | analog | analog | analog | analog | analog | analog |
| **Was?**<br>**Resultate** | analog | analog | analog | analog | analog | analog |

Neben der Verknüpfung mehrerer Tabellen (Kapitel 4 JOIN) ist die **WHERE**-Klausel der wichtigste Bestandteil des SELECT-Befehls. Je sorgfältiger die Auswahlbedingungen formuliert werden, desto genauer ist das Ergebnis der Abfrage.
Außerdem kann auch die **ORDER BY**-Klausel verwendet werden, eine Sortieranweisung. Das Ergebnis einer Anfrage soll entweder aufsteigend oder absteigend sortiert werden: **DESC** für DESCENDING, absteigend und **ASC** für ASCENDING, aufsteigend. Wird nach der ORDER BY-Klausel nicht explizit ASC oder DESC angegeben, so erfolgt die Sortierung immer aufsteigend. Gleiche Attribute in der Spalte können mit **SELECT DISTINCT** ausgeschlossen werden. DISTINCT gilt immer für den gesamten Attributvektor.

Mit den folgenden Vergleichsoperatoren können alle Datentypen verglichen werden – Zahlen, Datumsangaben, Zeichenketten.

| | |
|---|---|
| **Istgleich** | **=** |
| **Ungleich** | **!=** |
| **Kleiner** | **<** |
| **Größer** | **>** |
| **Kleiner als oder gleich** | **<=** |
| **Größer als oder gleich** | **>=** |

## 3.1 OPERATOR GLEICH

Der Operator **istgleich** sucht Datensätze, bei denen der Wert einer Spalte gleich einem vorgegebenen Wert ist.

Beispiel:

Die Attribute aller männlichen Kunden.

```
SELECT *
FROM kunde
WHERE Anrede = 'Herr';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Attribute der Kunden.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Kunden.

**3. Welcher Filter wird verwendet?**

Die männlichen Kunden werden über die Bedingung Anrede = 'Herr' gefiltert. (Satzfilter)

***1. Aufgabe:***

Alle Attribute der Aufträge mit dem Status 'offen'.

***2. Aufgabe:***

Artikel-ID, Farbe und Bezeichnung aller schwarzen Artikel.

## 3.2 OPERATOR UNGLEICH

Der Operator **ungleich** sucht Datensätze, bei denen der Wert einer Spalte ungleich einem vorgegebenen Wert ist.

Beispiel:

Alle Artikelgruppenbezeichnungen aus der Tabelle Artikelgruppe, bis auf Maus.

```
SELECT Artikelgrbez
FROM artikelgruppe
WHERE Artikelgrbez != 'Maus';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Artikelgruppenbezeichnungen.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikelgruppe.

**3. Welcher Filter wird verwendet?**

Die Artikelgruppenbezeichnungen werden über die Bedingung != 'Maus' gefiltert. (Satzfilter)

### *1. Aufgabe:*

Die Attribute  aller Aufträge bis auf die offenen Aufträge.

### *2. Aufgabe:*

Alle Farben der Artikel bis auf schwarz.

## 3.3 OPERATOR KLEINER

Der Operator **kleiner** sucht Datensätze, bei denen der Wert der Spalte in einer Tabelle kleiner als ein vorgegebener Wert ist.

Beispiel:

Alle Kunden-IDs, die kleiner sind als 'k3'.

```
SELECT KID
FROM kunde
WHERE KID < 'k3';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Kunden-IDs.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Kunde.

**3. Welcher Filter wird verwendet?**

Die KIDs werden über die Bedingung < 'k3' gefiltert. (Satzfilter)

***1. Aufgabe:***

Die Attribute aller Aufträge, die vor dem 09.10.2012 aufgenommen wurden.

***2. Aufgabe:***

Alle Artikelpreise, die günstiger sind als 150€.

## 3.4 OPERATOR GRÖßER

Der Operator **größer** sucht Datensätze, bei denen der Wert der Spalte in einer Tabelle größer als ein vorgegebener Wert ist.

Beispiel:

Alle Artikelbezeichnungen der Artikel, deren Artikelbezeichnung größer als 'R' ist.

```
SELECT Artikelbez
FROM artikel
WHERE Artikelbez > 'R';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Artikelbezeichnungen.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Die Artikelbezeichnungen werden über die Bedingung > 'R' gefiltert. (Satzfilter)

*1. Aufgabe:*

Alle Positionswerte der Auftragspositionen, deren Wert größer ist als 180€, absteigend nach Positionswert sortiert. Gleiche Beträge sollen nicht vorkommen.

*2. Aufgabe:*

PLZen, die größer sind als 80500 ausgeben. Geben Sie zu den PLZ der Kundenwohnorte auch die Kunden-IDs aus.

## 3.5 OPERATOR KLEINERGLEICH

Der Operator **kleinergleich** sucht Datensätze, bei denen der Wert der Spalte in einer Tabelle kleinergleich einem vorgegebenen Wert ist.

Beispiel:

Alle Auftrag-IDs, die kleiner oder gleich '2A' sind.

```
SELECT AID
FROM auftrag
WHERE AID <= '2A';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Auftrags-IDs.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Auftrag.

**3. Welcher Filter wird verwendet?**

Die Auftrags-IDs werden über die Bedingung <= '2A' gefiltert. (Satzfilter)

***1. Aufgabe:***

Alle Auftragspositionen mit ID und Positionsmenge, bei denen die Positionsmenge kleiner oder gleich zwei ist. Absteigend nach Positionsmenge sortieren.

***2. Aufgabe:***

Alle Attribute der Kunden, deren Name kleiner oder gleich 'L' ist.

## 3.6 OPERATOR GRÖßERGLEICH

Der Operator **größergleich** sucht Datensätze, bei denen der Wert der Spalte in einer Tabelle größer oder gleich einem vorgegebenen Wert ist.

Beispiel:

Alle Artikelpreise, die größer oder gleich 200€ sind.

```
SELECT Preis
FROM artikel
WHERE Preis >= 200;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Preise der Artikel.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Die Preise werden über die Bedingung >= 200 gefiltert. (Satzfilter)

***1. Aufgabe:***

Gewicht und IDs der Artikel, die mindestens 40kg wiegen.

***2. Aufgabe:***

Alle Attribute der Lieferanten, deren Name größer oder gleich 'H' ist.

## 3.7 BETWEEN AND – WERTE ZWISCHEN ZWEI GRENZEN

Ein Bereich wird mit der Bedingung **BETWEEN** <wert1> **AND** <wert2> verglichen. Die Grenzen gehören dazu. Diese Bedingung sucht Datensätze, bei denen der Betrag zwischen zwei Werten liegt.

Beispiel:

Die Preise aller Artikel, die zwischen 100€ und 200€ liegen.

```
SELECT Preis
FROM artikel
WHERE Preis BETWEEN 100 AND 200;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Preise der Artikel.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Die Preise der Artikel werden durch BETWEEN 100 AND 200 gefiltert. (Satzfilter)

***1. Aufgabe:***

ID und Auftragsdatum der Aufträge, deren Datum zwischen dem 2012-09-01 und 2012-12-31 liegt. Absteigend sortiert nach der Auftrags-ID.

***2. Aufgabe:***

IDs, Vornamen und Namen der Kunden, die im PLZ-Bereich 1000 und 80500 leben.

Auch eine Verneinung ist hier möglich. Mit **NOT BETWEEN AND** würde die Syntax der Aufgabe 2 alle PLZ wählen, die außerhalb des Bereichs 1000 und 80500 liegen. Auch hier gehören die Grenzen dazu.

## 3.8 LIKE

Die **LIKE**-Bedingung vergleicht Zeichenketten „ungenau“. Als Wert soll der gesuchte Text in einer Spalte enthalten sein. Dazu werden „Wildcards“ benutzt: Der Unterstrich '_' steht für ein beliebiges einzelnes Zeichen, das an der betreffenden Stelle vorankommen kann. Das '%' Zeichen steht für beliebige Zeichenketten mit keinem oder mehr Zeichen.

In zwei Situationen wird diese Bedingung gerne benutzt:

- Der Suchbegriff ist sehr lang; dem Anwender soll es genügen, den Anfang einzugeben.
- Der Suchbegriff ist nicht genau bekannt

1. Beispiel:

Artikel mit der Eigenschaft: Die Artikelbezeichnung enthält irgendwo 'an', der Inhalt dahinter und davor ist beliebig.

```
SELECT *
FROM artikel
WHERE Artikelbez LIKE '%an%';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Attribute der Artikel.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Die Artikelbezeichnungen werden über die Bedingung LIKE '%an%' gefiltert. (Satzfilter)

2. Beispiel:

Eine Verneinung ist hier mit **NOT LIKE** möglich. Artikel mit der Eigenschaft: Die Artikelbezeichnung beginnt nicht mit 'S', der Inhalt dahinter ist beliebig.

```
SELECT *
FROM artikel
WHERE Artikelbez NOT LIKE 'S%';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Attribute der Artikel.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Die Artikelbezeichnungen werden über die Bedingung NOT LIKE 'S%' gefiltert. (Satzfilter)

3. Beispiel:

Der Anfangsbuchstabe der Artikelbezeichnung ist unklar, aber danach folgen die Buchstaben 'ca' und eventuell noch etwas mehr.

```
SELECT *
FROM artikel
WHERE Artikelbez LIKE '_ca%';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Attribute der Artikel.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Die Artikelbezeichnungen werden über die Bedingung LIKE '_ca%' gefiltert. (Satzfilter)

Doch wenn ein Wort mit einem der Wildcard-Zeichen gesucht wird, muss dem LIKE-Operator mitgeteilt werden, dass '%' bzw. '_' als „echtes“ Zeichen zu verstehen ist.
Dies geschieht dadurch, dass ein spezielles Zeichen (Backslash) davor gesetzt wird:

4. Beispiel:

Artikelbezeichnungen mit Anfangsbuchstaben 'E' und einem Unterstrich ('_') dahinter, der Inhalt dahinter ist beliebig.

```
SELECT Artikelbez
FROM artikel
WHERE Artikelbez LIKE 'E%\_%';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Artikelbezeichnungen.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Die Artikelbezeichnungen werden über die Bedingung LIKE '%E_%' gefiltert. (Satzfilter)

Die Prozentzeichen stehen dafür, dass vorher und nachher beliebige Inhalte möglich sind. Der Unterstrich wird mit dem Escape-Zeichen '\' verbunden und ist damit Teil der gesuchten Zeichenfolge. Diese Angabe '\' ist als ein Zeichen zu verstehen.

***1. Aufgabe:***

Alle Attribute der Kunden, die in ihrem Namen irgendwo ein 'ei' enthalten.

***2. Aufgabe:***

IDs und Namen der Lieferanten, deren Name mit 'M' beginnt.

***3. Aufgabe:***

Der Anfangsbuchstabe der Kundenadresse ist unklar, aber danach folgen die Buchstaben 'an' und noch etwas mehr. Kunden-ID, Kundenname und Straße ausgeben.

***4. Aufgabe:***

IDs und Größen der Artikel, die irgendwo ein 'x' in der Größe enthalten.

## 3.9 IS NULL (NULL-WERTE PRÜFEN)

**IS NULL** bestimmt ob, ein angegebener Wert NULL, also leer, ist. NULL ist nicht die Zahl 0 und es ist auch keine leere Zeichenkette ' '. Wie im unteren Beispiel zu sehen ist, wird geprüft, ob ein Feld Einzelpreisrabatt in der Tabelle Auftragsposition leer ist, also keinen Einzelpreisrabatt enthält.
Der Test auf nicht NULL kann mit **IS NOT NULL** gemacht werden. Hier wird getestet, ob ein Attribut nicht leer ist.

Beispiel:
Alle Attribute der Auftragspositionen, bei denen der Einzelpreisrabatt den Wert NULL besitzt. Absteigend nach Positionsmenge sortieren.

```
SELECT *
FROM auftragsposition
WHERE Einzelpreisrabatt IS NULL
ORDER BY Positionsmenge DESC;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Attribute der Auftragspositionen.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Auftragsposition.

**3. Welcher Filter wird verwendet?**

Die Einzelpreisrabatte werden über den Operator IS NULL gefiltert. (Satzfilter)

*1. Aufgabe:*

Name und Vorname der Kunden, die einer Kundengruppe zugeordnet sind.

*2. Aufgabe:*

Alle Attribute der Aufträge, die keinem Mitarbeiter zugeordnet sind.

## 3.10 IN - VERGLEICH MIT EINER LISTE/SUBSELECT

Ermittelt, ob ein angegebener Wert mit einem Wert aus einer Unterabfrage oder Liste übereinstimmt. Eine Verneinung ist mit **NOT IN** möglich.

Beispiel **IN**:

Alle Kunden-IDs der Kunden, die in Deutschland oder in Österreich wohnen.

```
SELECT KID
FROM kunde
WHERE Land IN ('Deutschland', 'Österreich');
```

Kommentierung:

**1. Welche Attribute brauche ich?**

IDs und Länder der Kunden.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Kunde.

**3. Welcher Filter wird verwendet?**

Die Länder werden über die Bedingung IN ('Deutschland', 'Österreich') gefiltert. (Satzfilter)

Beispiel **NOT IN**:

Lieferanten, die keine Artikel geliefert haben, sollen mit ID und Name ausgegeben werden.

```
SELECT LID, Name
FROM lieferant
WHERE LID
NOT IN (SELECT L_LID
        FROM artikel);
```

Kommentierung:

**Innerer SELECT**

**1. Welche Attribute brauche ich?**

Lieferanten-ID.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Semantik:**

IDs der Lieferanten, die schon Artikel geliefert haben.

**Äußerer SELECT**

**1. Welche Attribute brauche ich?**

Lieferanten-ID und -name.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Lieferant.

**3. Welcher Filter wird verwendet?**

Die Lieferanten-ID wird über den Operator NOT IN gefiltert.

***1. Aufgabe:***

Alle IDs, Namen und Orte von Kunden, die in Erlangen und in München leben.

***2. Aufgabe:***

IDs und Städte der Standorte, die in der Tabelle Abteilung vorkommen.

***3. Aufgabe:***

IDs und Namen der Mitarbeiter, die noch keinen Auftrag bearbeitet haben.

| | Kapitel 4.1 | Kapitel 4.2 | Kapitel 4.3 | Kapitel 4.4 |
|---|---|---|---|---|
| **Woher?**<br>**Ausgangspunkt**<br>**Problemstellung** | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **INNER JOIN**s erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **LEFT OUTER JOIN**s erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **negativen JOIN**s erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **RIGHT OUTER JOIN**s erlernen und üben können. |
| **Wohin?**<br>**Zieldefinition** | Es soll eine Anleitung erstellt werden, die die Problemstellung löst. | analog | analog | analog |
| **Wie?**<br>**Methoden (-verwendung)** | Der INNER JOIN ist theoretisch und anhand von Beispielen zu erklären. Übungsaufgaben für eine bestimmte An wesentlicher Befehlsvarianten sind zu entwerfen. | analog | analog | analog |
| **Was?**<br>**Resultate** | Diese Anleitung enthält die JOIN-Variante: INNER JOIN. Die Variante wird theoretisch erklärt, und dazu werden zwei Beispiele mit Kommentierung angegeben. Für die Studierenden sind zwei Übungsbeispiele angegeben, die sie selbstständig lösen und kommentieren sollen. | analog | analog | analog |

| | Kapitel 4.5 | Kapitel 4.6 | Kapitel 4.7 |
|---|---|---|---|
| **Woher?**<br>**Ausgangspunkt**<br>**Problemstellung** | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **FULL OUTER JOIN**s erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **CROSS JOIN**s erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des **SELF JOIN**s erlernen können. |
| **Wohin?**<br>**Zieldefinition** | analog | analog | analog |
| **Wie?**<br>**Methoden (-verwendung)** | analog | analog | analog |
| **Was?**<br>**Resultate** | analog | Diese Anleitung enthält die JOIN-Variante: CROSS JOIN. Die Variante wird theoretisch erklärt, und dazu werden zwei Beispiele mit Kommentierung angegeben. | Diese Anleitung enthält die JOIN-Variante: SELF JOIN. Die Variante wird theoretisch erklärt, und dazu wird ein Beispiel mit Kommentierung angegeben. Für die Studierenden sind zwei Übungsbeispiele angegeben, die sie selbstständig lösen und kommentieren sollen. |

## 4.1 INNER JOIN (KOMMUTATIV)

Unter einem **JOIN** versteht man einen Verbund von Relationen, d.h. ein Kreuzprodukt mit Nebenbedingung. Dies bedeutet, dass die Ergebnismenge sich aus Spalten verschiedener Tabellen zusammensetzt. In der SQL-Syntax kommt es auf die Reihenfolge der betroffenen Tabellen nicht an. Angenommen, es soll ausgegeben werden, welcher Mitarbeiter (Name) mit welchem Kunden (Name) einen Auftrag abgewickelt hat. Da diese Informationen sich in diesem Fall über drei Tabellen verteilen, ist eine Abfrage über alle beteiligten Tabellen notwendig. Zu je zwei Tabellen muss jeweils mindestens eine Spalte mit gleicher Semantik vorhanden sein.
Mit **AS** kann man einer Tabelle einen neuen Namen geben.

Kartesisches Produkt in der Mengenlehre: $A \times B = \{(a, b) \mid a \in A \text{ und } b \in B\}$

verallgemeinerte Produktbildung für n Mengen ($n \geq 2$): $A_1 \times \ldots \times A_n = \{(a_1, \ldots, a_n) \mid a_i \in A_i \mid 1 \leq i \leq n\}$

Beispiel:

**$A \times B$ der Mengen A = (a, b, c, d) und B = (3, 5, 7):**

**$A \times B$ = { (a,3), (a,5), (a,7), (b,3), (b,5), (b,7), (c,3), (c,5), (c,7), (d,3), (d,5), (d,7)}**

| | 3 | 5 | 7 |
|---|---|---|---|
| a | (a,3) | (a,5) | (a,7) |
| b | (b,3) | (b,5) | (b,7) |
| c | (c,3) | (c,5) | (c,7) |
| d | (d,3) | (d,5) | (d,7) |

**Abbildung 26: Kartesisches Produkt**

1. Beispiel:

Zusammengehörige Kundennamen (umbenennen in „Kunde“), Auftrags-IDs, Mitarbeiternamen (umbenennen in „Mitarbeiter“) der Tabellen Kunde, Auftrag und Mitarbeiter.

*(Die Tabellenkürzel schließen Redundanzen aus, d.h. sie können angegeben werden, sind aber nicht erforderlich, wenn keine gleichen Attribute in den jeweiligen Tabellen vorkommen.)*

```
SELECT k.Name AS Kunde, [a.]AID, m.Name AS Mitarbeiter
FROM kunde k JOIN (
auftrag [a] JOIN mitarbeiter m ON [a.]M_MID = [m.]MID)
ON [k.]KID = [a.]K_KID;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Kundenname, Auftrags-ID und Mitarbeitername.

**2. Aus welchen Tabellen kommen die Attribute?**

Tabellen Kunde, Auftrag und Mitarbeiter.

2. Beispiel:

Alle Auftrags- und Auftragspositions-IDs sowie Einzelpreisrabatte, wobei der Einzelpreisrabatt größer oder gleich 5% ist.

```
SELECT AID, APID, Einzelpreisrabatt
FROM auftrag
JOIN auftragsposition ON AID = A_AID
WHERE Einzelpreisrabatt >= 5;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Auftrags-ID, Auftragspositions-ID und Einzelpreisrabatt.

**2. Aus welchen Tabellen kommen die Attribute?**

Tabellen Auftrag und Auftragsposition.

**3. Welcher Filter wird verwendet?**

Der Einzelpreisrabatt wird über die Bedingung >= 5 gefiltert. (Satzfilter)

***1. Aufgabe:***

Alle Auftrags-IDs und die zugehörige Kunden mit Namen (umbenennen in „Kundenname“) ausgeben, die einen Einzelpreisrabatt bekommen haben. Absteigend nach Kundenname sortieren.

***2. Aufgabe:***

Alle Lieferanten von Artikeln mit der Artikelgruppenbezeichnung „Scanner“.

## 4.2 LEFT OUTER JOIN (NICHT KOMMUTATIV)

Wenn eine Abfrage eine Verknüpfung enthält, wird die Tabelle vor dem Schlüsselwort **JOIN** als **„linke Tabelle“** bezeichnet. Analog dazu wird die Tabelle hinter dem Schlüsselwort als „rechte Tabelle" bezeichnet. Die linke äußere Verknüpfung (LEFT OUTER JOIN) übernimmt die Zeilen der linken Tabelle in jedem Fall in das Ergebnis. Wenn ein Datensatz der rechten Tabelle dem ON-Kriterium entspricht, so werden die zur rechten Tabelle gehörenden Spalten entsprechend befüllt, ansonsten bleiben sie leer (NULL).

1. Beispiel: **Kontrollabfrage:**

Testen, ob alle Artikelgruppen Artikel enthalten. Artikelgruppen- und Artikel-IDs sowie Artikelgruppenbezeichnungen ausgeben und aufsteigend nach der Artikelgruppen-ID sortieren.

```
SELECT AGID, ARID, Artikelgrbez
FROM artikelgruppe
LEFT JOIN artikel ON AGID = AG_AGID
ORDER BY AGID ASC;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Artikelgruppen-ID, Artikel-ID und Artikelgruppenbezeichnung.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabellen Artikelgruppe und Artikel.

**Alternativbefehl:**

```
SELECT AGID, ARID, Artikelgrbez
FROM artikel JOIN
artikelgruppe ON AGID = AG_AGID
UNION
SELECT AGID, " ", Artikelgrbez
FROM artikelgruppe
WHERE AGID NOT IN(
SELECT DISTINCT AG_AGID
FROM artikel
WHERE AG_AGID IS NOT NULL)
ORDER BY AGID ASC;
```

**Semantik:**
alle Kombinationen AGID, ARID, Artikelgruppenbezeichnung

**Semantik:**
Artikelgruppen-IDs der Artikelgruppen, denen keine Artikel zugeordnet sind.

**Semantik (SELECT):**
AGIDs der Artikelgruppen, denen Artikel zugeordnet sind

**Kommentar zum 2. SELECT:**

Das Leerzeichen (" ") wird dazugeschrieben, damit beide Mengen die gleiche Struktur haben.

2. Beispiel: **Kontrollabfrage:**

Testen, ob alle Artikel einer Artikelgruppe zugeordnet sind. Artikel- und Artikelgruppen-IDs sowie Artikelgruppenbezeichnungen ausgeben und aufsteigend nach der Artikel-ID sortieren.

```
SELECT ARID, AGID, Artikelgrbez
FROM artikel
LEFT JOIN artikelgruppe ON AGID = AG_AGID
ORDER BY ARID ASC;
```

**Alternativbefehl: Artikel mit und ohne Artikelgruppe**

```
SELECT ARID, AGID, Artikelgrbez
FROM artikelgruppe
JOIN artikel ON AGID = AG_AGID
UNION
SELECT ARID, " ", " "
FROM artikel
WHERE AG_AGID IS NULL
ORDER BY ARID ASC;
```

**Semantik:**
alle Kombinationen ARID, AGID
Artikelgruppenbezeichnung

**Semantik:**
Artikel-IDs der Artikel, denen keine
Artikelgruppen zugeordnet sind

**Kommentar zum 2. SELECT:**

Die Leerzeichen (" ") werden dazugeschrieben, damit beide Mengen die gleiche Struktur haben.

***1. Aufgabe:***

Geben Sie alle Abteilungsbezeichnungen und die zugehörigen Mitarbeiternamen und Abteilungs-IDs aus der Tabelle Mitarbeiter aus. Aufsteigende Sortierung nach Abteilungsbezeichnung.

***2. Aufgabe:***

Alle Auftragspositions-IDs mit und ohne Einzelpreisrabatt ausgeben. Geben Sie auch die Kunden-IDs und Namen der Kunden aus und sortieren Sie nach der Auftragspositions-ID.

## 4.3 NEGATIVER JOIN

**LEFT JOIN mit WHERE-Bedingung** verknüpft zuerst die Tabellen und filtert dann in der WHERE-Klausel die Bedingung. Während die andere Variante, LEFT JOIN mit der **Bedingung im ON-Kriterium,** einen INNER JOIN aus dem LEFT JOIN macht.

Beispiel:

Geben Sie alle Artikel der Artikelgruppen aus, deren Bezeichnung mit 'A' beginnt, mit der zugehörigen Artikelgruppe. Geben Sie auch alle anderen Artikelgruppen an.

**LEFT JOIN mit WHERE-Bedingung:**

```
SELECT AGID, Artikelgrbez, ARID
FROM artikelgruppe
LEFT JOIN artikel ON AGID = AG_AGID
WHERE Artikelgrbez LIKE 'L%';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Artikelgruppen-ID, Artikelgruppenbezeichnung und Artikel-ID

**2. Aus welchen Tabellen kommen die Attribute?**

Tabelle Artikelgruppe und Artikel.

**3. Welcher Filter wird verwendet?**

Die Artikelgruppenbezeichnung wird über die Bedingung LIKE 'L%' gefiltert. (Satzfilter)

**LEFT JOIN: Bedingung im ON-Kriterium:**

```
SELECT AGID, Artikelgrbez, ARID
FROM artikelgruppe
LEFT JOIN artikel ON AGID = AG_AGID
AND Artikelgrbez LIKE 'L%';
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Artikelgruppen-ID, Artikelgruppenbezeichnung und Artikel-ID

**2. Aus welchen Tabellen kommen die Attribute?**

Tabelle Artikelgruppe und Artikel.

**3. Welcher Filter wird verwendet?**

Es wird mit der mit AND verbundenen Bedingung LIKE 'L%' gefiltert. (Satzfilter)

***1.Aufgabe:***

Geben Sie alle Artikel der Artikelgruppen, deren Bezeichnung mit 'L' oder 'M' beginnt,  aus.

***2. Aufgabe:***

Geben Sie alle Mitarbeiter ohne Abteilung aus.

## 4.4 RIGHT OUTER JOIN

Die rechte äußere Verknüpfung (RIGHT OUTER JOIN) funktioniert analog. Sie übernimmt Zeilen aus der rechten Tabelle in jedem Fall in das Ergebnis. Wenn ein Datensatz der linken Tabelle dem ON-Kriterium entspricht, so werden die zur linken Tabelle gehörenden Spalten entsprechend befüllt, ansonsten bleiben sie leer (NULL).

Beispiel:

Liste aller Standorte mit ID und Stadt sowie Abteilungsbezeichnung ausgeben. Auch Abteilungen, die noch keinem Standort zugeordnet sind aufsteigend sortiert nach der Standort-ID ausgeben.

```
SELECT SID, Stadt, Bezeichnung
FROM standort
RIGHT OUTER JOIN abteilung ON S_SID = SID
ORDER BY SID;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Standort-ID, Stadt und Abteilungsbezeichnung.

**2. Aus welchen Tabellen kommen die Attribute?**

Tabellen Standort und Abteilung.

***1. Aufgabe:***

Kundenname (umbenennen in „Kundenname“) mit und ohne Kundengruppenbezeichnung (umbenennen in „Kundengruppe“) ausgeben.

***2. Aufgabe:***

Auftragsstatus und Positionswerte sowie Kundennamen (umbenennen in „Kundenname“) ausgeben. Auch Aufträge ohne Status sollen ausgegeben werden.

## 4.5 FULL OUTER JOIN

Die vollständige äußere Verknüpfung (FULL OUTER JOIN) kombiniert die Funktion der linken und rechten äußeren Verknüpfung. Sie übernimmt aus der linken und der rechten Tabelle alle Datensätze in das Ergebnis. Findet sich über das ON-Kriterium ein passender Partner werden beide zusammengefügt, andernfalls wird die jeweils fehlende Seite mit NULL aufgefüllt.
Leider unterstützt MySQL keinen FULL OUTER JOIN. Um dennoch so eine Abfrage zu erzeugen, muss ein LEFT JOIN und ein RIGHT JOIN per UNION verbunden werden.

Beispiel: **Motivation zur Fehlersuche**

Alle Abteilungen mit und ohne Mitarbeiter und alle Mitarbeiter mit und ohne Abteilung absteigend sortiert nach der Mitarbeiter-ID ausgeben.

```
SELECT ABID, MID
FROM abteilung
LEFT JOIN mitarbeiter ON (ABID = AB_ABID)
UNION
SELECT ABID, MID
FROM abteilung
RIGHT JOIN mitarbeiter ON (ABID = AB_ABID)
ORDER BY MID DESC;
```

Kommentierung:

**1. SELECT-Teil:**

**1. Welche Attribute brauche ich?**

Mitarbeiter- und Abteilungs-ID.

**2. Aus welchen Tabellen kommen die Attribute?**

Tabellen Abteilung und Mitarbeiter.

**3. Semantik:**

Alle Abteilungs-IDs ggf. mit Mitarbeiter-ID.

**2. SELECT-Teil:**

**1. Welche Attribute brauche ich?**

Mitarbeiter- und Abteilungs-ID.

**2. Aus welchen Tabellen kommen die Attribute?**

Tabellen Abteilung und Mitarbeiter.

**3. Semantik:**

Alle Mitarbeiter-IDs ggf. mit Abteilungs-ID.

**Alternativbefehl:**

```
SELECT MID, ABID
FROM mitarbeiter
JOIN abteilung ON ABID = AB_ABID
UNION
SELECT " ", ABID
FROM abteilung
WHERE ABID NOT
IN (SELECT DISTINCT AB_ABID
    FROM mitarbeiter
    WHERE AB_ABID IS NOT NULL)
UNION
SELECT MID, " "
FROM mitarbeiter
WHERE AB_ABID IS NULL
ORDER BY MID DESC;
```

**Semantik:**
alle Kombinationen MID, ABID

**Semantik:**
Abteilungs-IDs der Abteilungen, denen keine Mitarbeiter zugeordnet sind

**Semantik (SELECT):**
Abteilungs-IDs der Abteilungen, denen Mitarbeiter zugeordnet sind

**Semantik:**
Mitarbeiter-IDs der Mitarbeiter, denen keine Abteilungen zugeordnet sind

**Kommentar zum 2. Und 3. SELECT:**

Die Leerzeichen (" ") werden dazugeschrieben, damit beide Mengen die gleiche Struktur haben.

*1. Aufgabe:*

Kunden-IDs, Einzelpreisrabatt, Auftragsstatus für Kunden-IDs mit und ohne Einzelpreisrabatt und Kunden-IDs mit und ohne Auftragsstatus.

*2. Aufgabe:*

Alle Kundengruppen mit und ohne Kunden und alle Kunden mit und ohne Kundengruppen.

## 4.6 CROSS JOIN (KARTESISCHES PRODUKT)

Verbindet jede Zeile der ersten Tabelle mit jeder Zeile der zweiten Tabelle. CROSS-JOIN ist die einfache Verknüpfung ohne WHERE-Klausel. Deshalb kann die Anweisung

```
SELECT *
FROM kunde, auftrag;
```

auch folgendermaßen geschrieben werden:

```
SELECT *
FROM kunde CROSS JOIN auftrag;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Attribute der Kunden und der Aufträge.

**2. Aus welchen Tabellen kommen die Attribute?**

Tabellen Kunde und Auftrag.

Das Ergebnis ist ein kartesisches Produkt (auch Kreuzprodukt genannt) der beiden Ausgangstabellen. Leider kann mit diesem Konstrukt, wie oben gezeigt wurde, kaum etwas angefangen werden, da das Ergebnis eines **CROSS JOIN**s häufig nutzlos ist und bei entsprechender Datensatzauswahl jede Datenbank abstürzen lässt.

## 4.7 SELF-JOIN

Beim **SELF-JOIN** werden nicht zwei verschiedene Tabellen benutzt, sondern zweimal dieselbe Tabelle.

Beispiel:

Mitarbeiter-ID, Mitarbeitername (mit Umbenennung) und Vorgesetzten-Name (mit Umbenennung) ausgeben.

```
SELECT m.MID, m.Name AS Mitarbeiter, v.Name AS Vorgesetzter
FROM mitarbeiter m JOIN mitarbeiter v
ON v.MID = m.M_Vorgesetzter;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Mitarbeiter-ID, Mitarbeitername, Vorgesetztenname.

**2. Aus welcher Tabelle kommen die Attribute?**

Zweimal die Tabelle Mitarbeiter.

*1. Aufgabe:*

Mitarbeiter (umbenennen in „Mitarbeiter“) und Vorgesetzter (umbenennen in „Vorgesetzter), die in der gleichen Abteilung (umbenennen in „Abteilung“) tätig sind.

*2. Aufgabe:*

Mitarbeiter (mit Umbenennung), die mindestens einen Auftrag (Auftrags-ID) bearbeitet haben, mit dem Vorgesetzten (mit Umbenennung) ausgeben.

| | Kapitel 5.1 | Kapitel 5.2 | Kapitel 5.3 |
|---|---|---|---|
| **Woher?**<br>Ausgangspunkt<br>Problemstellung | Es fehlt eine Anleitung, zur Darstellung von Mengen. Studierende sollen im Selbststudium die Verwendung von Mengenlehre in Bezug auf SQL erlernen und üben können. | Es fehlt eine Anleitung, zur Darstellung von Logik. Studierende sollen im Selbststudium die Verwendung von Logik in Bezug auf SQL erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung der Mengenvereinigung in Bezug auf SQL erlernen und üben können. |
| **Wohin?**<br>Zieldefinition | Es soll eine Anleitung erstellt werden, die die Problemstellung löst. | analog | analog |
| **Wie?**<br>Methoden<br>(-verwendung) | Mengendefinitionen werden theoretisch erklärt. | Logik wird theoretisch erklärt, und die Junktoren werden vorgestellt. | Die Mengenvereinigung ist theoretisch und anhand von Beispielen zu erklären. Übungsaufgaben für eine bestimmte Anzahl wesentlicher Befehlsvarianten sind zu entwerfen. |
| **Was?**<br>Resultate | In diesem Abschnitt werden die Darstellungsformen explizit und implizit erklärt. | Dieser Abschnitt des Kapitels enthält die Junktoren:<br>$\vee$, $\wedge$ und $\neg$. | Die Mengenvereinigung wird theoretisch erklärt, und dazu wird ein Beispiel mit Kommentierung angegeben. Für die Studierenden sind zwei Übungsbeispiele angegeben, die sie selbstständig lösen und kommentieren sollen. |

| | Kapitel 5.4 | Kapitel 5.5 | Kapitel 5.6 |
|---|---|---|---|
| **Woher?**<br>Ausgangspunkt<br>Problemstellung | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung des Mengendurchschnitts in Bezug auf SQL erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung von Differenzmenge und Mengenkomplement in Bezug auf SQL erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung der Symmetrischen Differenz in Bezug auf SQL erlernen und üben können. |
| **Wohin?**<br>Zieldefinition | analog | analog | analog |
| **Wie?**<br>Methoden<br>(-verwendung) | analog | analog | analog |
| **Was?**<br>Resultate | analog | analog | analog |

## 5.1 Darstellung von Mengen

Mengen werden meistens mit Großbuchstaben dargestellt und die Elemente mit Kleinbuchstaben.

| |
|---|
| a ist ein Element von M: $a \in M$<br>ist a nicht Element von M: $a \notin M$ |

Es gibt zwei Formen der Darstellung von Mengen:

**1. explizit:** Aufzählung aller Elemente der Menge: $\{a_1, a_2, a_3, \ldots\}$

**2. implizit:** Menge X mittels einer Eigenschaft : $\{x \mid E(x)\}$

**Beispiele:**

1. {2, 4, 6, 8} ist die Menge der Zahlen 2, 4, 6 und 8
2. {x | x ist eine gerade und natürliche Zahl und x<10}

Eine Menge, die kein Element enthält, heißt leere Menge und wird durch die Notation $\varnothing$ dargestellt.
Bei expliziter Mengendarstellung spielt die Reihenfolge keine Rolle. Mengenelemente haben an sich keine Reihenfolge. Die Mengen {1, 2, 3} und {3, 2, 1} sind gleich. Auch die Anzahl des Auftretens eines Elements in einer Mengenrepräsentation ist irrelevant. Mengen enthalten jedes ihrer Elemente nur einmal. Die Menge {1, 2, 2} stimmt mit der Menge {1, 2} überein.
Die Mengenlehre kennt drei grundlegende Operationen, mit denen man zwei Mengen verknüpfen kann. **Vereinigung**[12], **Durchschnitt**[13] **und Differenzmenge**[14].

## 5.2 Logik allgemein

Zusammengesetzte Aussagen werden mit Hilfe von Junktoren abgebildet.

**Zweistellige Junktoren** (lat Junktor = Verbinder) der Aussagelogik:

| | | | |
|---|---|---|---|
| Konjunktion | $\wedge$ | „und“ | (auch: &) |
| Disjunktion | $\vee$ | „oder“ | (auch: \|) |

**Einstelliger Junktor:**

| | | |
|---|---|---|
| Negation | ¬ | „nicht“ |

[12] Kapitel 5.3 Vereinigung
[13] Kapitel 5.4 Durchschnitt
[14] Kapitel 5.5 Differenzmenge und Komplement

## 5.3 VEREINIGUNG (KOMMUTATIV)

Die **Vereinigung** zweier Mengen ist die Menge, die diejenigen Elemente enthält, die wenigstens in einer der beiden Mengen enthalten ist, sie umfasst also die Elemente beider Mengen.
Sind **A** und **B** zwei Mengen, so existiert die Vereinigung (oder Vereinigungsmenge) von **A** und **B**:

$$A \cup B = \{ x \mid x \in A \vee x \in B \}$$

Beispiel: $\{1, 2\} \cup \{2, 3\} = \{1, 2, 3\}$

Aufgabe:
Kundennamen und Auftragspositions-IDs der Auftragspositionen, bei denen die Positionsmenge größer 2 ist oder der Einzelpreisrabatt über 5% liegt oder der Positionswert über 200€ beträgt. Absteigend nach Auftragspositions-ID sortieren.

```
SELECT Name, APID
FROM auftragsposition JOIN (
kunde JOIN
auftrag ON KID=K_KID) ON AID = A_AID
WHERE Positionsmenge > 2
OR Einzelpreisrabatt > 5
OR Positionswert > 200
ORDER BY APID DESC;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Kundenname, Auftragspositions-ID, Positionsmenge, Einzelpreisrabatt und Positionswert .

**2. Aus welchen Tabellen kommen die Attribute?**

Tabelle Auftragsposition, Kunde und Auftrag.

**3. Welcher Filter wird verwendet?**

Es wird mit den mit OR verbundenen Bedingungen Positionsmenge > 2, Einzelpreisrabatt > 5 und Positionswert > 200 gefiltert. (Satzfilter)

**Alternativbefehl:**

Der SQL-Befehl ***UNION ALL*** vereinigt ebenfalls die Ergebnismengen zweier Abfragen. Der Unterschied zwischen ***UNION ALL*** und ***UNION*** besteht darin, dass mit ***UNION*** nur unterschiedliche Werte ausgewählt werden, während bei ***UNION ALL*** alle Werte, also auch mehrfach vorkommende Ergebnistupel erhalten bleiben.

***1. Variante mit Dubletten:***

```
SELECT Name, APID
FROM auftragsposition JOIN (
kunde JOIN auftrag ON KID = K_KID) ON AID = A_AID
WHERE Positionsmenge > 2
UNION ALL
SELECT Name, APID
FROM auftragsposition JOIN (
kunde JOIN auftrag ON KID = K_KID) ON AID = A_AID
WHERE Positionswert > 200
UNION ALL
SELECT Name, APID
FROM auftragsposition JOIN (
kunde JOIN auftrag ON KID = K_KID) ON AID = A_AID
WHERE Einzelpreisrabatt > 5
ORDER BY APID DESC;
```

***1. Variante ohne Dubletten:***

```
SELECT Name, APID
FROM auftragsposition JOIN (
kunde JOIN auftrag ON KID = K_KID) ON AID = A_AID
WHERE Positionsmenge > 2
UNION
SELECT Name, APID
FROM auftragsposition JOIN (
kunde JOIN auftrag ON KID = K_KID) ON AID = A_AID
WHERE Positionswert > 200
UNION
SELECT Name, APID
FROM auftragsposition JOIN (
kunde JOIN auftrag ON KID = K_KID) ON AID = A_AID
WHERE Einzelpreisrabatt > 5
ORDER BY APID DESC;
```

***Für die Bearbeitung der Aufgaben wird UNION verwendet:***

***UNION*** wirkt im Wesentlichen wie ein **ODER**-Operator. Auswahl von Werten, die entweder in der ersten oder in der zweiten Anweisung vorkommen.

***1. Aufgabe:***

Auftragsstatus, Kundengruppenbezeichnung und alle Attribute der Kunden von den Aufträgen, deren Status 'offen' oder die Kundengruppenbezeichnung der Kunden 'C' ist.

***2. Aufgabe:***

Artikel-ID (mit Umbenennung), Lieferantenname (mit Umbenennung), Artikelfarbe und Artikelpreis von den Artikeln, die vom Lieferanten mit der ID 'L2' geliefert werden oder deren Artikelfarbe schwarz ist oder deren Preis über 40€ liegt.

## 5.4 DURCHSCHNITT (KOMMUTATIV)

Der **Durchschnitt** zweier Mengen **A** und **B** ist als diejenige Menge definiert, die alle Elemente enthält, die in beiden Mengen vorhanden sind:

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

Beispiel: $\{1, 2\} \cap \{2, 3\} = \{2\}$

Aufgabe:

Alle Artikelbezeichnungen der Artikel, deren Farbe schwarz ist und deren Preis über 100€ liegt. Absteigend nach Artikelbezeichnung sortieren.

```
SELECT Artikelbez
FROM artikel
WHERE Farbe = 'schwarz'
AND Preis > 100
ORDER BY Artikelbez DESC;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Artikelbezeichnung, Farbe und Preis.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Es wird mit den mit AND verbundenen Bedingungen Farbe = 'schwarz' und Preis > 100 gefiltert. (Satzfilter)

**Alternativbefehl:**

```
SELECT Artikelbez
FROM artikel
WHERE Farbe = 'schwarz'
INTERSECT
SELECT Artikelbez
FROM artikel
WHERE Preis > 100
ORDER BY Artikelbez DESC;
```

***Für die Bearbeitung der Aufgaben wird INTERSECT verwendet:***

Ähnlich wie der Befehl ***UNION*** bezieht sich auch ***INTERSECT*** auf zwei SQL-Anweisungen. Der Befehl ***INTERSECT*** wirkt wie ein **UND**-Operator und betrachtet die Auswahl von Werten, die in beiden Anweisungen vorkommen.

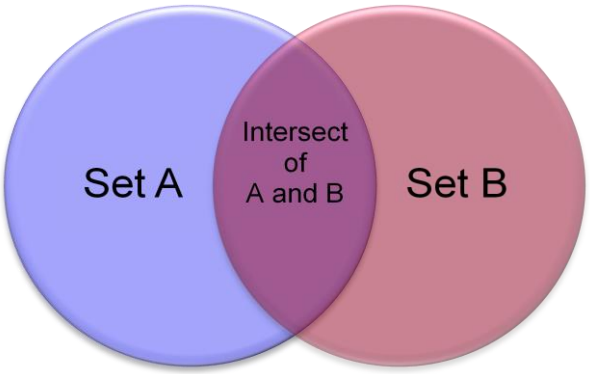


**Abbildung 27: INTERSECT**

***1. Aufgabe:***

Positionsmenge und Einzelpreisrabatt der Auftragspositionen, bei denen die Menge größer 3 und der Einzelpreisrabatt größer oder gleich 5% ist. Aufsteigend nach Positionsmenge sortieren.

***2. Aufgabe:***

Auftragsstatus und Lieferdatum aller Aufträge, die vom Mitarbeiter Graf bearbeitet wurden und den Status 'geliefert' haben.

## 5.5 Differenzmenge und Komplement (nicht kommutativ)

Die **Differenzmenge** zweier Mengen enthält alle Elemente, die in der ersten Menge **A** enthalten sind und nicht in der zweiten Menge **B**.

$$A \setminus B = \{x \mid x \in A \text{ und } x \notin B\}$$

**Beispiel:** **{1, 2} \ {2, 3} ={1}**

Die Schreibweise für das Komplement von B bzgl. A ist $B_A^c$ (ausgesprochen: „Komplement von B“).
Ist die Menge A als Grundmenge vorausgesetzt und B eine Teilmenge von A, wird vom Komplement der Menge B gesprochen:

$$B_A^c = \{x \in A \mid x \notin B : A \supset B\}$$

Unter Negation einer Aussage A versteht man die Aussage ¬A (in Worten: „nicht A“), die genau dann wahr ist, wenn A falsch ist. Die Verneinung ist etwas anderes als das umgangssprachliche Gegenteil. Die Verneinung von „x ist schwarz“ ist keineswegs die Aussage „x ist weiß“. Denn „nicht schwarz“ könnte auch z. B. „rot“ oder „blau“ bedeuten.

**Beispiel:** **{1, 2, 3} \ {2, 3} ={1}**

Aufgabe (Differenzmenge):
Alle Attribute der Auftragspositionen, die einen Einzelpreisrabatt unter 10% haben und deren Positionsmenge nicht kleiner als 5 ist.

```
SELECT *
FROM auftragsposition
WHERE Einzelpreisrabatt < 10
AND NOT Positionsmenge < 5;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Alle Attribute der Auftragspositionen.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Auftragsposition.

**3. Welcher Filter wird verwendet?**

Es wird mit den mit AND NOT verbundenen Bedingungen Einzelpreisrabatt < 10 und Positionsmenge < 5 gefiltert. (Satzfilter)

**Alternativbefehl:**

```
SELECT *
FROM auftragsposition
WHERE Einzelpreisrabatt < 10
MINUS
SELECT *
FROM auftragsposition
WHERE Positionsmenge < 5;
```

***Für die Bearbeitung der Aufgaben wird MINUS/EXCEPT verwendet:***

Der Befehl ***MINUS/ EXCEPT*** wirkt auf zwei SQL-Anweisungen. Er schließt für die Antwort von allen Ergebnissen der ersten SQL-Anweisung diejenigen aus, die auch in der zweiten SQL-Anweisung enthalten sind. Enthält die zweite SQL-Anweisung Ergebnisse, die in der ersten nicht vorhanden sind, so werden diese ignoriert.

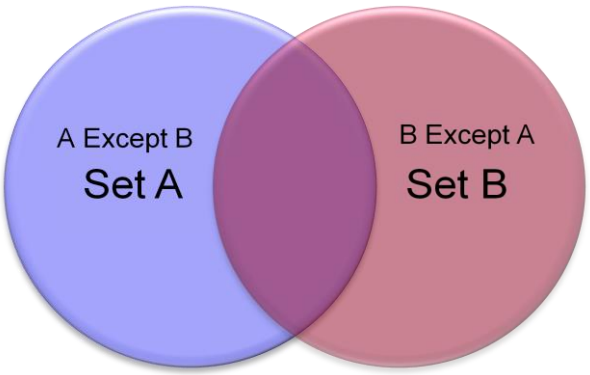


**Abbildung 28: EXCEPT**

***1. Aufgabe:***

Abteilungsbezeichnungen und Städte, wobei die Stadt nicht Köln und nicht Berlin ist.

***2. Aufgabe:***

Geben Sie für alle männlichen Kunden den Namen und die zugehörigen Aufträge mit Auftragspositions-ID und Auftragspositionswert aus. Es sollen nur Ergebnisse mit einem Positionswert größer oder gleich 150€ ausgegeben werden.

## 5.6 Symmetrische Differenz (kommutativ)

Die **symmetrische Differenz** zweier Mengen enthält genau diejenigen Elemente, die in exakt einer der beiden Mengen enthalten sind.
Exklusives, ausschließendes ODER: Entweder **A** oder **B**, aber nicht beides:

$$A \Delta B = (A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B) = \{x \mid ((x \in A) \wedge (x \notin B)) \vee ((x \in B) \wedge (x \notin A))\}$$

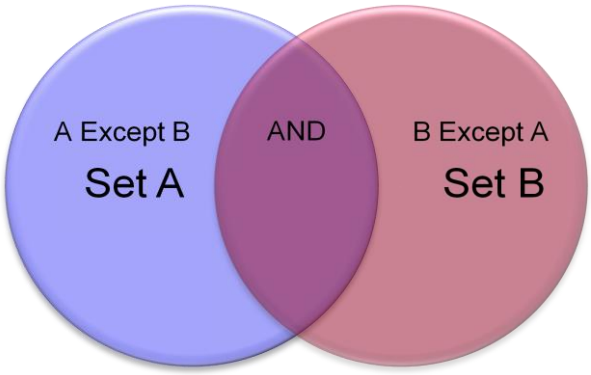


**Abbildung 29: Venn-Diagramm Symmetrische Differenz**

Beispiel: $A \Delta B = (\{1, 2\} \setminus \{2, 3\}) \cup (\{2, 3\} \setminus \{1, 2\}) = \{1\} \cup \{3\} = \{1, 3\}$

Aufgabe:
Artikel-ID, Artikelbezeichnung und Artikelpreis: Entweder mit der Artikelbezeichnung 'A' bis 'R' oder Artikelpreis zwischen 100€ und 200€. Aufsteigend nach Artikelbezeichnung sortiert.

```
SELECT ARID, Artikelbez, Preis
FROM artikel
WHERE Artikelbez < 'R'
XOR Preis BETWEEN 100 AND 200
ORDER BY Artikelbez;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Artikel-ID, Artikelbezeichnung und Artikelpreis.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

**3. Welcher Filter wird verwendet?**

Es wird mit den mit XOR verbundenen Bedingungen Artikelbez < 'R' und Preis BETWEEN 100 AND 200 gefiltert. (Satzfilter)

***1. Aufgabe:***

Status und Auftragsdatum der Aufträge, die entweder am 2012-10-05 erstellt wurden oder den Status 'offen' haben.

***2. Aufgabe:***

Artikelpreis sowie Positionswert und Positionsmenge der Auftragspositionen, bei denen entweder die Positionsmenge größer 3 und der Positionswert größer 40€ ist oder der Artikelpreis größer 150€ ist.

| | Kapitel 6.1 | Kapitel 6.2 | Kapitel 6.3 |
|---|---|---|---|
| **Woher?**<br>**Ausgangspunkt**<br>**Problemstellung** | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung von Gruppenbildung und Aggregatfunktion **SUM()** erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung von Gruppenbildung und Aggregatfunktion **COUNT()** erlernen und üben können. | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung der Gruppenbildung und Aggregatfunktion **AVG()** erlernen und üben können. |
| **Wohin?**<br>**Zieldefinition** | Es soll eine Anleitung erstellt werden, die die Problemstellung löst. | analog | analog |
| **Wie?**<br>**Methoden**<br>**(-verwendung)** | Die Aggregatfunktion ist theoretisch und anhand von Beispielen zu erklären. Übungsaufgaben für eine bestimmte Anzahl wesentlicher Befehlsvarianten sind zu entwerfen. | analog | analog |
| **Was?**<br>**Resultate** | Die Aggregatfunktionsvariante SUM wird theoretisch erklärt, und dazu wird ein Beispiel mit Kommentierung angegeben. Für die Studierenden sind zwei Übungsbeispiele angegeben, die sie selbstständig lösen und kommentieren sollen. | analog | analog |

| | **Kapitel 6.4** |
|---|---|
| **Woher?**<br>**Ausgangspunkt**<br>**Problemstellung** | Es fehlt eine Anleitung, mit der die Studierenden im Selbststudium die Verwendung von Gruppenbildung und Aggregatfunktionen **MAX()** und **MIN()** erlernen und üben können. |
| **Wohin?**<br>**Zieldefinition** | analog |
| **Wie?**<br>**Methoden (-verwendung)** | analog |
| **Was?**<br>**Resultate** | analog |

Neben dem **WHERE-Filter** gibt es noch den **HAVING-Filter.** WHERE ist ein Satzfilter und HAVING ein Gruppenfilter. Der Unterschied ist, dass sich die Reihenfolge bei der **GROUP BY**-Klausel beim HAVING und WHERE ändern, d. h. die Gruppierung erfolgt bei HAVING vor der HAVING-Filterung und beim WHERE danach.

Oftmals werden nur einzelne Datensätze aus den Tabellen zurückgegeben. In SQL ist es aber möglich, die Datensätze einer Tabelle in Gruppen zusammenzufassen und dann die Aggregatfunktionen jeweils auf die Gruppen anzuwenden. Die Aggregatfunktionen beziehen sich entweder auf die gesamte Selektion einer Anfrage, die eine einzige Gruppe bildet oder auf Gruppen von Datensätzen.

Es gibt die fünf Aggregatfunktionen **SUM(), COUNT(), AVG(), MAX(), MIN()**.

## *6.1 SUM()*

Die Aggregatfunktion **SUM()** summiert die Werte einer Spalte je Gruppe.

Beispiel:

Auftrags-ID und Summe der Positionswerte (umbenennen in „Auftragssumme“) je Auftrag. Die Ausgabe soll nach der Auftragssumme absteigend sortiert werden.

```
SELECT A_AID, SUM(Positionswert) AS Auftragssumme
FROM auftragsposition
GROUP BY A_AID
ORDER BY Auftragssumme DESC;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Auftrags-ID und Positionswert.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Auftragsposition.

**3. Wie wird gruppiert?**

Es wird nach Auftrags-ID gruppiert.

***1. Aufgabe:***

Positionsmengen aller Auftragspositionen summieren (umbenennen in „Artikelmenge“) und nach Artikel-ID gruppieren. Aufsteigende Sortierung nach Artikelmenge.

***2. Aufgabe:***

Positionswerte der jeweiligen Auftragspositionen summieren (umbenennen in „Summe pro Kunde“) und nach Kunden-ID gruppieren.

## 6.2 COUNT()

Die Aggregatfunktion **COUNT()** listet die Anzahl der Datensätze pro Gruppe auf.
**COUNT(*)** dahingehend gibt die Anzahl der abgerufenen Datensätze an, unabhängig davon ob diese NULL-Werte enthalten oder nicht.

Beispiel:
Anzahl der offenen und Anzahl der gelieferten Aufträge.

```
SELECT COUNT(Status) AS Anzahl_Aufträge, Status
FROM auftrag
WHERE Status IN ('offen', 'geliefert')
GROUP BY Status;
```

Kommentierung:

**1 Welche Attribute brauche ich?**

Status.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Auftrag.

**3. Welcher Filter wird verwendet?**

Es wird über die Bedingung IN ('offen', 'geliefert') gefiltert. (Satzfilter)

**4. Wie wird gruppiert?**

Es wird nach Status gruppiert.

**Alternativbefehl mit HAVING:**

```
SELECT COUNT(Status) AS Anzahl_Aufträge, Status
FROM auftrag
GROUP BY Status
HAVING Status IN ('offen', 'geliefert');
```

***1. Aufgabe:***

Artikelbezeichnung (umbenennen in „Artikelname“) und Anzahl der Kunden (umbenennen in „Anzahl Kunde“), die einen schwarzen Artikel bestellt haben. Gruppiert nach Artikelname.

***2. Aufgabe:***

Alle Auftrags-IDs, mit mehr als einer Auftragsposition ausgeben. Gruppiert und absteigend sortiert nach Auftrags-ID.

***3. Aufgabe:***

Anzahl der Auftragspositionen (umbenennen in „Anzahl der Auftragspositionen“) je Kunden-ID. Aufsteigend nach der Anzahl der Auftragspositionen sortieren.

***4. Aufgabe:***

Wie viele unterschiedliche Einzelpreisrabatte gibt es in der Tabelle Auftragsposition? Tabelle umbenennen in „Anzahl Einzelpreisrabatte“.

## 6.3 AVG()

Die Aggregatfunktion **AVG()** berechnet pro Gruppe die Durchschnittswerte.

Beispiel:

Durchschnittspreis der Artikel.

```
SELECT AVG(Preis) AS "Durchschnittspreis"
FROM artikel;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Artikelpreis.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Artikel.

***1. Aufgabe:***

Der durchschnittliche Positionswert der Einkäufe des Kunden Schneider (umbenennen in: „Kunde Schneider“).

***2. Aufgabe:***

Auftrags-ID und durchschnittlicher Einzelpreisrabatt je Auftrag, wobei nur durchschnittliche Einzelpreisrabatte angezeigt werden sollen, die größer als 5% sind. Absteigende Sortierung nach durchschnittlichem Einzelpreisrabatt.

## 6.4 MAX() UND MIN()

Die Aggregatfunktionen **MAX()** und **MIN()** geben pro Gruppe die Maximal- und Minimalwerte aus.

Beispiel:

Der höchste und der niedrigste Positionswert.

```
SELECT MAX(Positionswert) AS höchster_Wert,
MIN(Positionswert) AS niedrigster_Wert
FROM auftragsposition;
```

Kommentierung:

**1. Welche Attribute brauche ich?**

Positionswert.

**2. Aus welcher Tabelle kommen die Attribute?**

Tabelle Auftragsposition.

***1. Aufgabe:***

Kunden-IDs und jeweils das erste Auftragsdatum (umbenennen in „Erster Auftrag“) der Kunden. Aufsteigende Sortierung nach Kunden-ID.

***2. Aufgabe:***

Der jeweils letzte Auftrag pro Mitarbeiter. Aufsteigende Sortierung nach Mitarbeiter-ID.

***3. Aufgabe:***

Die maximale und die minimale Positionsmenge je Auftrag. Absteigende Sortierung nach Auftrags-ID.

***4. Aufgabe:***

Den höchsten Positionswert eines Auftrags und den zugehörigen Bearbeiter dieses Auftrags ausgeben. Aufsteigende Sortierung nach Mitarbeiter-ID.

| | Kapitel 7.1 | Kapitel 7.2 |
|---|---|---|
| **Woher?**<br>**Ausgangspunkt**<br>**Problemstellung** | Es fehlt ein Datenmodell, anhand dessen die Studierenden im Selbststudium echt schwierige Aufgaben erlernen und üben können. | Es fehlt ein Abschnitt, mit dem die Studierenden im Selbststudium echt schwierige Aufgaben erlernen und üben können. |
| **Wohin?**<br>**Zieldefinition** | Es soll ein Abschnitt einer Anleitung erstellt werden, der die Problemstellung löst. | analog |
| **Wie?**<br>**Methoden (-verwendung)** | Es wird ein neues komplexes Datenmodell erstellt. | Zu diesem Datenmodell sind schwierige Aufgaben definiert, die Themen der anderen Kapitel vertiefen. |
| **Was?**<br>**Resultate** | Das neue Datenmodell „Vorlesung“ wurde mit Hilfe von Microsoft Visio erstellt. | Dieser Abschnitt der Anleitung enthält schwierigere Aufgaben als in den anderen Kapiteln. Für die Studierenden sind Übungsaufgaben angegeben, die sie selbständig lösen und kommentieren sollen. |

## 7.1 Datenmodell „Vorlesung“

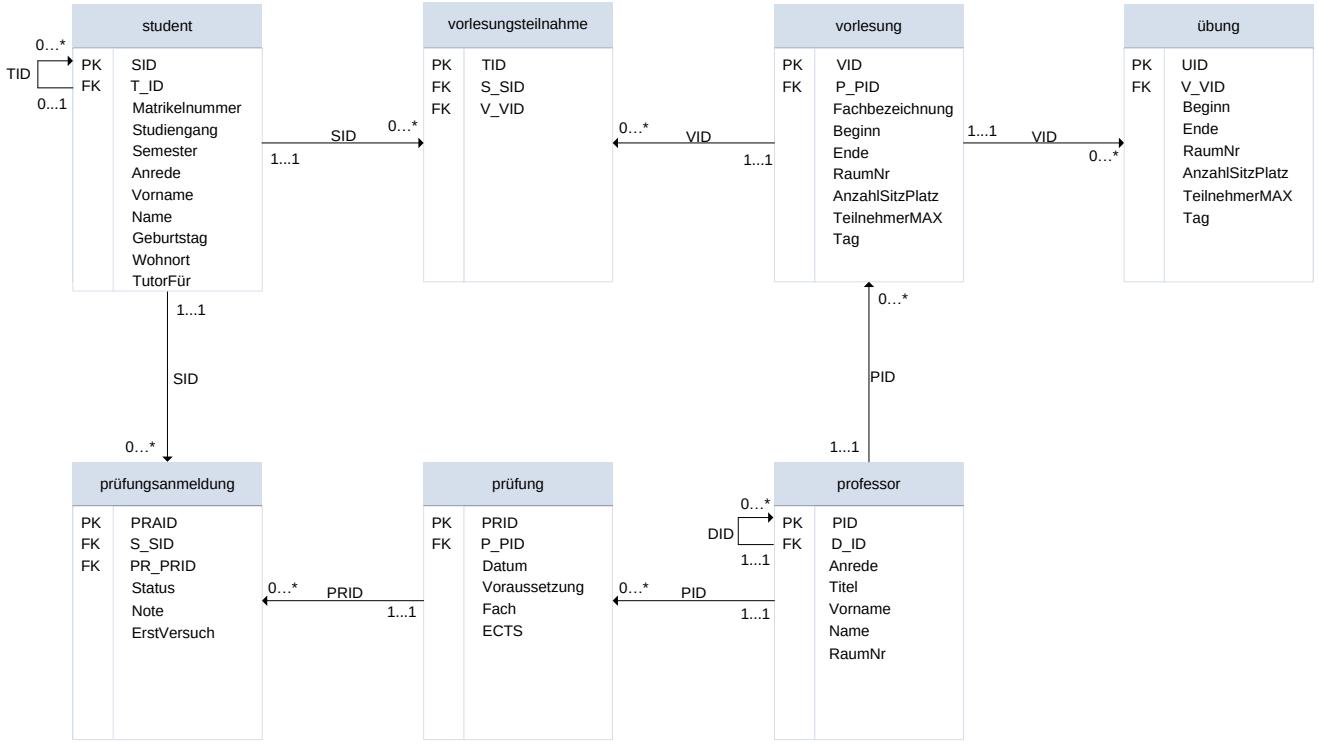


**Abbildung 30: Datenmodell „Vorlesung“**

Kommentar:

**n:m Student – Vorlesung wird aufgelöst:**

Ein **Student** kann an keiner oder mehreren Vorlesungen teilnehmen.

Eine **Vorlesungsteilnahme** ist genau einem Studenten zugewiesen.

Eine **Vorlesung** kann keine oder mehrere Vorlesungsteilnahmen haben, und eine Vorlesungsteilnahme gehört genau einer Vorlesung.

Ein **Student** kann ein **Tutor** sein oder er kann einen bzw. keinen Tutor haben.

Eine **Vorlesung** kann keine oder mehrere Übungen haben.

Eine **Übung** gehört genau einer Vorlesung.

Jeder Student, der sich für die Vorlesungsteilnahme anmeldet, muss die Übung als Voraussetzung für die Prüfung besuchen und wird deshalb auch automatisch zur Übung angemeldet.

Eine **Vorlesung** wird von genau einem Professor gehalten.

Ein **Professor** kann keine oder mehrere Vorlesungen halten.

Ein Professor kann ein **Dekan** sein.

Ein **Professor** kann keine oder mehrere Prüfungen erstellen.

Eine **Prüfung** wird von genau einem Professor erstellt.

**n:m Student – Prüfung wird aufgelöst:**

Ein **Student** kann sich für keine oder mehrere Prüfungen anmelden.

Eine **Prüfungsanmeldung** wir genau einem Studenten zugeordnet.

Eine **Prüfung** kann keine oder mehrere Prüfungsanmeldungen enthalten, und eine Prüfungsanmeldung ist genau einer Prüfung zugeordnet.

**student**

| SID | TU_ID | Matrikelnummer | Studiengang | Semester | Anrede | Vorname | Name | Geburtstag | Wohnort | TutorFür |
|---|---|---|---|---|---|---|---|---|---|---|
| S100 | S200 [->] | 2222 | Informatik | 3 | Herr | Markus | Müller | 1992-01-20 | Erlangen | *NULL* |
| S200 | *NULL* | 1111 | Wirtschaftsinformatik | 4 | Herr | Daniel | Meier | 1991-06-10 | Nürnberg | Software-Engineering |
| S300 | S200 [->] | 3333 | Wirtschaftsinformatik | *NULL* | Frau | Sandra | Schulze | 1989-12-01 | Fürth | *NULL* |
| S400 | *NULL* | 4444 | Informatik | 5 | Herr | Julian | Neumann | 1991-06-10 | Nürnberg | Datenbanken |
| S500 | *NULL* | 5555 | Informatik | 6 | Herr | Simon | Koch | 1988-05-10 | Nürnberg | Datenbanken |
| S600 | S500 [->] | *NULL* | *NULL* | 1 | Frau | Anna | Wagner | *NULL* | Fürth | *NULL* |
| S700 | S500 [->] | 6666 | Informatik | 6 | Frau | Selina | Lütke | 1990-05-12 | Roth | *NULL* |
| S800 | *NULL* | 1234 | Informatik | 7 | Frau | Sabine | Schiller | 1988-02-21 | Schwabach | *NULL* |

**vorlesungsteilnahme**

| TID | S_SID | V_VID |
|---|---|---|
| T3 | S300 [->] | *NULL* |
| T6 | S200 [->] | V1 [->] |
| T9 | S700 [->] | V1 [->] |
| T2 | S400 [->] | V1 [->] |
| T13 | S500 [->] | V2 [->] |
| T7 | S400 [->] | V2 [->] |
| T5 | S100 [->] | V2 [->] |
| T8 | S600 [->] | V3 [->] |
| T12 | S100 [->] | V4 [->] |
| T10 | S500 [->] | V4 [->] |
| T1 | S700 [->] | V4 [->] |
| T11 | S100 [->] | V5 [->] |
| T4 | S700 [->] | V5 [->] |
| T14 | S600 [->] | V5 [->] |

**vorlesung**

| VID | P_PID | Fachbezeichnung | Beginn | Ende | RaumNr | AnzahlSitzPlatz | TeilnehmerMax | Tag |
|---|---|---|---|---|---|---|---|---|
| V1 | P1 [->] | Datenbanken | 08:00:00 | 09:30:00 | V101 | 100 | 78 | Montag |
| V2 | P2 [->] | Software-Engineering | 09:45:00 | 10:30:00 | V105 | 150 | 66 | Freitag |
| V3 | P3 [->] | Englisch | 08:00:00 | 10:30:00 | V101 | 100 | 89 | Mittwoch |
| V4 | P2 [->] | MEntU | 11:15:00 | 13:00:00 | *NULL* | *NULL* | 30 | Montag |
| V5 | P3 [->] | Statistik | 08:00:00 | 09:00:00 | *NULL* | *NULL* | 0 | Mittwoch |
| V6 | *NULL* | Mathematik I | 08:00:00 | 10:30:00 | V104 | *NULL* | 0 | Donnerstag |
| V7 | *NULL* | TI | 09:45:00 | 11:15:00 | *NULL* | *NULL* | 0 | Dienstag |
| V8 | *NULL* | Physik | *NULL* | *NULL* | *NULL* | *NULL* | 20 | Dienstag |

**übung**

| UID | V_VID | Beginn | Ende | AnzahlSitzPlatz | RaumNr | TeilnehmerMAX | Tag |
|---|---|---|---|---|---|---|---|
| U-V0 | V7 [->] | 08:00:00 | 08:45:00 | *NULL* | *NULL* | 15 | Freitag |
| U-V1 | V1 [->] | 08:00:00 | 09:30:00 | 40 | V306 | 40 | Dienstag |
| U-V2 | V2 [->] | 14:00:00 | 15:00:00 | 30 | V305 | 30 | Donnerstag |
| U-V3 | V3 [->] | 11:30:00 | 13:00:00 | *NULL* | *NULL* | 0 | Montag |

**professor**

| PID | D_ID | Anrede | Titel | Vorname | Name | RaumNr |
|---|---|---|---|---|---|---|
| P1 | P2 [->] | Frau | Prof. Dr. | Claudia | Graf | K300 |
| P2 | *NULL* | Herr | Prof. Dr. | Norbert | Gerstner | K200 |
| P3 | P2 [->] | Herr | *NULL* | Reinhard | Bauer | K100 |
| P4 | P2 [->] | Frau | Dr. | Susanne | Stamminger | *NULL* |
| P5 | *NULL* | Herr | *NULL* | Thomas | Bindler | *NULL* |

**prüfung**

| PRID | P_PID | Datum | Voraussetzung | Fach | ECTS |
|---|---|---|---|---|---|
| PR-1 | P1 [->] | 2013-07-18 | Übung | Datenbanken | 5 |
| PR-3 | P3 [->] | 2013-07-10 | *NULL* | Englisch | 2 |
| PR-4 | *NULL* | 2013-07-15 | *NULL* | MEntU | 6 |
| PR-2 | P2 [->] | 2013-07-22 | Übung | Software-Engineering | 5 |
| PR-5 | *NULL* | 2013-07-09 | *NULL* | Statistik | 5 |

**prüfungsanmeldung**

| PRAID | S_SID | PR_PRID | Status | Note | ErstVersuch |
|---|---|---|---|---|---|
| PA-1 | S100 [->] | PR-5 [->] | angemeldet | 4.0 | *NULL* |
| PA-2 | S100 [->] | PR-2 [->] | angemeldet | *NULL* | *NULL* |
| PA-3 | S200 [->] | PR-1 [->] | angemeldet | 3.0 | *NULL* |
| PA-5 | S400 [->] | PR-2 [->] | angemeldet | *NULL* | *NULL* |
| PA-4 | S400 [->] | PR-1 [->] | angemeldet | 1.7 | 5 |
| PA-6 | S500 [->] | PR-4 [->] | angemeldet | *NULL* | 5 |
| PA-7 | S600 [->] | PR-3 [->] | abgeschlossen | 2.3 | *NULL* |
| PA-10 | S700 [->] | PR-5 [->] | angemeldet | 5.0 | *NULL* |
| PA-8 | S700 [->] | PR-1 [->] | angemeldet | 5.0 | 5 |
| PA-9 | S700 [->] | PR-4 [->] | *NULL* | *NULL* | 5 |

**Abbildung 31: Testdaten**

## 7.2 Aufgaben

# STUDENT

**Aufgabe 1**

Ermitteln Sie den Vornamen, Namen und das Alter des jüngsten Studenten und der jüngsten Studentin.

**Aufgabe 2**

Wie viele verschiedene Werte für das Alter und für den Geburtstag treten bei den Studenten auf? Sortieren Sie nach der Anzahl der verschiedenen Geburtstage.

**Aufgabe 3**

Bestimmen Sie für alle Studenten, die 21 sind, den Vornamen, Namen, Studiengang und das Alter. Geben Sie dazu die Übungen an, die sie besuchen. Sortieren Sie aufsteigend nach Namen.

**Aufgabe 4**

Wie viele Studenten mit dem Studiengang Informatik nehmen an der Vorlesung Statistik teil?

**Aufgabe 5**

Studenten die entweder im 4. Semester studieren oder ein Tutor in einem beliebigen Fach sind. Wie viele Vorlesungen besuchen diese Studenten?

**Aufgabe 6**

Ermitteln Sie alle Tutoren (mit Umbenennung), die an Vorlesungen teilnehmen. Geben Sie die Fachbezeichnung an und sortieren Sie aufsteigend nach der Studenten-ID.

**Aufgabe 7 a**

Geben Sie eine SQL-Anweisung an, mit der die IDs aller Studenten angezeigt werden, die **genau** eine Vorlesung bei dem Professor mit dem Namen „Gerstner“ hören. Sortieren Sie absteigend nach der Studenten-ID.

**Aufgabe 7 b**

Geben Sie eine SQL-Anweisung an, mit der die IDs aller Studenten angezeigt werden, die **mindestens** eine Vorlesung bei dem Professor mit dem Namen „Gerstner“ hören. Geben Sie auch die Fachbezeichnung und die Dauer der Vorlesung (umbenennen in „Vorlesungsdauer“) an. Sortieren Sie aufsteigend nach der Studenten-ID.

**Aufgabe 8**

Ermitteln Sie für alle Studenten, die „Wirtschaftsinformatik“ studieren und an der Vorlesung Datenbanken teilnehmen, die ID und die Matrikelnummer. Sortieren Sie das Ergebnis aufsteigend nach der Matrikelnummer.

**Aufgabe 9**

Ermitteln Sie die durchschnittliche Note (umbenennen in „Durchschnitts_Note“) aller Fächer für alle Studenten. Runden Sie die Noten auf ganze Zahlen. Geben Sie dabei nur diejenigen Studenten an, bei denen die Durchschnittsnote kleiner als 3 ist.

**Aufgabe 10**

Studenten, die noch keine Prüfung belegt haben, sollen aufgelistet werden. Aufsteigende Sortierung nach Studenten-ID.

**Aufgabe 11**

Geben Sie die Studenten mit ID aus, die im ersten Versuch einer beliebigen Prüfung eine 5 hatten. Sortieren Sie aufsteigend nach der Studenten-ID.

**Aufgabe 12 a**

Alle Studenten, die **mindestens** eine Prüfung bestanden haben. Geben Sie die Studenten-ID, das Prüfungsfach, den Status der Prüfungsanmeldung und die Note (umbenennen in „mit Erfolg“) aus und sortieren Sie aufsteigend nach der Note.

**Aufgabe 12 b**

Alle Studenten, die **alle** Prüfungen beim ersten Versuch bestanden haben. Geben Sie die Studenten-ID, das Prüfungsfach und den Status der Prüfungsanmeldung an und sortieren Sie aufsteigend nach der Note.

**Aufgabe 13**

Listen Sie alle Studenten aus Nürnberg auf, deren Note in den Prüfungsfächern zwischen 2 und 4 liegt.

**Aufgabe 14**

Geben Sie die Studenten mit ID, Prüfungsnote und Prüfungsfach aufsteigend sortiert nach dem Prüfungsfach aus. Geben Sie dazu den Professor an, der die jeweilige Klausur erstellt hat. Wenn der Ersteller der Prüfung noch nicht eingetragen wurde, soll die Liste trotzdem aufgelistet werden und für den Professor der Wert „Null“ stehen.

**Aufgabe 15 a**

Erstellen Sie eine SQL-Abfrage, die für jeden Professor die Studenten-IDs ausgibt, die an seinen Vorlesungen teilnehmen. Studenten, die keine Vorlesung besuchen, sollen auch ausgegeben werden. Geben Sie auch die Fachbezeichnung der Vorlesungen an. Aufsteigende Sortierung nach der Fachbezeichnung.

**Aufgabe 15 b**

Geben Sie eine SQL-Anweisung an, mit der die Anzahl der bereits angemeldeten Vorlesungsteilnehmer (umbenennen in „aktuelle Teilnehmerzahl“) pro Vorlesung angezeigt werden. Geben Sie auch die Fachbezeichnung sowie den Namen des jeweiligen Professors an, der die Vorlesung hält und sortieren Sie absteigend nach der Teilnehmeranzahl.

## VORLESUNG

**Aufgabe 16**

Erstellen Sie eine SQL-Abfrage, die die Namen aller Vorlesungen absteigend sortiert ausgibt zusammen mit dem Namen des Professors, der die Vorlesung hält. Professoren, denen keine Vorlesung zugeordnet ist, sollen auch ausgegeben werden.

**Aufgabe 17 a**

Ermitteln Sie die Anzahl der Übungen (umbenennen in „Anzahl der Übungen“), die mindestens 45 Minuten dauern. Sortieren Sie aufsteigend nach der Dauer.

**Aufgabe 17 b**

Ermitteln Sie alle Übungen, die mindestens 45 Minuten dauern („umbenennen in „Dauer“). Geben Sie auch das Übungsfach (umbenennen in „Übung“) an und sortieren Sie aufsteigend nach der Dauer.

**Aufgabe 17 c**

Gruppieren Sie alle Vorlesungen nach der Vorlesungsdauer (umbenennen in „Vorlesungsdauer“). Geben Sie die Anzahl aller Vorlesungen aus (umbenennen in „Anzahl der Vorlesungen“), die die gleiche Vorlesungsdauer haben. Sortieren Sie das Ergebnis absteigend nach der Vorlesungsdauer.

**Aufgabe 18**

Geben Sie alle aktuellen Vorlesungen, die um 08:00 Uhr beginnen, mit Beginn, Ende, Vorlesungsdauer (umbenennen in „Dauer“), RaumNr, Professor, Fachbezeichnung und Tag an. Es sollen nur Vorlesungen ausgegeben werden, die einem Vorlesungsraum zugeordnet sind. Wenn eine Vorlesung keinem Professor zugeordnet ist, soll die Fachbezeichnung trotzdem angezeigt werden.

# PROFESSOR

**Aufgabe 19**

Es gibt Professoren die entweder eine oder mehrere Vorlesungen halten. Studenten können zwar an mehreren Vorlesungen eines Professors teilnehmen, doch es sollen nur Professoren-IDs aufgelistet werden, in deren Vorlesungen die Studenten nur einmal vorkommen. Aufsteigende Sortierung nach der Professoren-ID.

# PRÜFUNG

**Aufgabe 20**

Geben Sie für alle bestandenen Prüfungen das Prüfungsfach, die Durchschnittsnote (umbenennen in „Durchschnittsnote“) und die beste jemals erzielte Note (umbenennen in „Beste Note“), sortiert nach der Durchschnittsnote, aus. Es sollen nur Durchschnittsnoten ausgegeben werden, die besser sind als 3,5.

## 3.1 OPERATOR GLEICH

### 1. Aufgabe

```
SELECT *
FROM auftrag
WHERE Status = 'offen';
```

### 2. Aufgabe

```
SELECT ARID, Farbe, Artikelbez
FROM artikel
WHERE Farbe = 'schwarz';
```

## 3.2 OPERATOR UNGLEICH

### 1. Aufgabe

```
SELECT *
FROM auftrag
WHERE Status != 'offen';
```

### 2. Aufgabe

```
SELECT Farbe
FROM artikel
WHERE Farbe != 'schwarz';
```

## 3.3 OPERATOR KLEINER

### 1. Aufgabe

```
SELECT *
FROM auftrag
WHERE AuftragsDatum < '2012-10-09';
```

### 2. Aufgabe

```
SELECT Preis
FROM artikel
WHERE Preis < 150;
```

## 3.4 Operator größer

### 1. Aufgabe

```
SELECT DISTINCT Positionswert
FROM auftragsposition
WHERE Positionswert > 180
ORDER BY Positionswert DESC;
```

### 2. Aufgabe

```
SELECT PLZ, KID
FROM kunde
WHERE PLZ > '80500';
```

## 3.5 Operator kleinergleich

### 1. Aufgabe

```
SELECT APID, Positionsmenge
FROM auftragsposition
WHERE Positionsmenge <= 2
ORDER BY Positionsmenge DESC;
```

### 2. Aufgabe

```
SELECT *
FROM kunde
WHERE Name <= 'L';
```

## 3.6 Operator größergleich

### 1. Aufgabe

```
SELECT ARID, Gewicht
FROM artikel
WHERE Gewicht >= 40;
```

### 2. Aufgabe

```
SELECT *
FROM lieferant
WHERE Name >= 'H';
```

## 3.7 BETWEEN AND – Werte zwischen zwei Grenzen

### 1. Aufgabe

```
SELECT AID, Auftragsdatum
FROM auftrag
WHERE AuftragsDatum
BETWEEN '2012-09-01'
AND '2012-12-31'
ORDER BY AID DESC;
```

### 2. Aufgabe

```
SELECT KID, Vorname, Name
FROM kunde
WHERE PLZ
BETWEEN '1000'
AND '80500';
```

## 3.8 LIKE

### 1. Aufgabe

```
SELECT *
FROM kunde
WHERE Name LIKE '%ei%';
```

### 2. Aufgabe

```
SELECT LID, Name
FROM lieferant
WHERE Name LIKE 'M%';
```

### 3. Aufgabe

```
SELECT KID, Name, Straße
FROM kunde
WHERE Straße LIKE '_an%';
```

### 4. Aufgabe

```
SELECT ARID, Groeße
FROM artikel
WHERE Groeße LIKE '%x%';
```

## 3.9 IS NULL (NULL-Werte prüfen)

### 1. Aufgabe

```
SELECT Name, Vorname
FROM kunde
WHERE KG_KGID IS NOT NULL;
```

### 2. Auftrag

```
SELECT *
FROM auftrag
WHERE M_MID IS NULL;
```

## 3.10 IN - Vergleich mit einer Liste/Subselect

### 1. Aufgabe

```
SELECT KID, Name, Ort
FROM kunde
WHERE Ort
IN ('München', 'Erlangen');
```

### 2. Aufgabe

```
SELECT SID, Stadt
FROM standort
WHERE SID
      IN (SELECT S_SID
      FROM abteilung);
```

### 3. Aufgabe

```
SELECT MID, Name
FROM mitarbeiter
WHERE MID NOT
      IN (SELECT M_MID
      FROM auftrag);
```

## 4.1 INNER JOIN (KOMMUTATIV)

### 1. Aufgabe

```
SELECT DISTINCT AID, Name AS Kundenname
FROM kunde
JOIN (auftrag
JOIN auftragsposition ON AID = A_AID) ON KID = K_KID
WHERE Einzelpreisrabatt IS NOT NULL
ORDER BY Name DESC;
```

### 2. Aufgabe

```
SELECT Name, Artikelgrbez
FROM lieferant
JOIN (artikel
JOIN artikelgruppe ON AGID = AG_AGID) ON LID = L_LID
WHERE artikelgrbez = 'Scanner';
```

## 4.2 LEFT OUTER JOIN (NICHT KOMMUTATIV)

### 1. Aufgabe

```
SELECT Bezeichnung, AB_ABID, Name AS Mitarbeiter
FROM abteilung
LEFT JOIN mitarbeiter ON (AB_ABID = ABID)
ORDER BY Bezeichnung;
```

**Alternativbefehl:**

```
SELECT Bezeichnung, AB_ABID, Name AS Mitarbeiter
FROM abteilung
JOIN mitarbeiter ON (AB_ABID = ABID)
UNION
SELECT Bezeichnung, " ", " "
FROM abteilung
WHERE ABID NOT
IN (SELECT DISTINCT AB_ABID
    FROM mitarbeiter
    WHERE AB_ABID IS NOT NULL)
ORDER BY Bezeichnung ASC;
```

## 2. Aufgabe

```
SELECT APID, KID, Name, Einzelpreisrabatt
FROM kunde
LEFT JOIN (auftragsposition
JOIN auftrag ON AID = A_AID) ON (KID = K_KID)
ORDER BY APID ASC;
```

### Alternativbefehl:

```
SELECT APID, KID, Name, Einzelpreisrabatt
FROM kunde
JOIN (auftragsposition a
JOIN auftrag ON AID = A_AID) ON (KID = K_KID)
UNION
SELECT " ", KID, Name, " "
FROM kunde
WHERE KID NOT
IN (SELECT K_KID
    FROM auftrag)
ORDER BY APID ASC;
```

## 4.3 Negativer JOIN

### 1. Aufgabe

**LEFT JOIN mit WHERE-Bedingung:**

```
SELECT AGID, Artikelgrbez, ARID
FROM artikelgruppe
LEFT JOIN artikel ON AGID = AG_AGID
WHERE Artikelgrbez LIKE 'L%'
OR Artikelgrbez LIKE 'M%';
```

**LEFT JOIN: Bedingung im ON-Kriterium:**

```
SELECT AGID, Artikelgrbez, ARID
FROM artikelgruppe
LEFT JOIN artikel ON AGID = AG_AGID
AND Artikelgrbez LIKE 'L%'
OR Artikelgrbez LIKE 'M%';
```

### 2. Aufgabe

**LEFT JOIN mit WHERE-Bedingung:**

```
SELECT MID, ABID
FROM mitarbeiter LEFT JOIN abteilung on ABID = AB_ABID
WHERE AB_ABID IS NULL;
```

**LEFT JOIN: Bedingung im ON-Kriterium:**

```
SELECT MID, ABID
FROM mitarbeiter
LEFT JOIN abteilung ON ABID = AB_ABID
AND AB_ABID IS NULL;
```

## 4.4 RIGHT OUTER JOIN

### 1. Aufgabe

```
SELECT Name AS Kundenname, KundengrBez AS Kundengruppe
FROM kunde
RIGHT JOIN kundengruppe ON KGID = KG_KGID
ORDER BY Kundengruppe ASC;
```

**Alternativbefehl:**

```
SELECT Name AS Kundenname, KundengrBez AS Kundengruppe
FROM kunde
JOIN kundengruppe ON KGID = KG_KGID
UNION
SELECT " ", KundengrBez
FROM kundengruppe
WHERE KGID NOT
IN (SELECT KG_KGID
    FROM kunde
    WHERE KG_KGID IS NOT NULL)
ORDER BY Kundengruppe ASC;
```

### 2. Aufgabe

```
SELECT Status, Positionswert, Name AS Kundenname
FROM kunde
RIGHT JOIN (auftrag
JOIN auftragsposition ON AID = A_AID)
ON KID = K_KID;
```

**Alternativbefehl:**

```
SELECT Status, Positionswert, Name AS Kundenname
FROM kunde
JOIN (auftrag
JOIN auftragsposition ON AID = A_AID)
ON KID = K_KID
UNION
SELECT " ", NULL, Name AS Kundenname
FROM kunde
WHERE KID NOT
IN (SELECT K_KID
    FROM auftrag
     WHERE K_KID IS NOT NULL);
```

## 4.5 Full Outer JOIN

### 1. Aufgabe

```
SELECT KID, Einzelpreisrabatt, Status
FROM kunde FULL JOIN (auftrag
JOIN auftragsposition ON AID = A_AID) ON KID = K_KID;
```

**Alternativbefehl:**

```
SELECT KID, Einzelpreisrabatt, Status
FROM kunde
LEFT JOIN (auftrag
JOIN auftragsposition ON AID = A_AID) ON KID = K_KID
UNION
SELECT KID, Einzelpreisrabatt, Status
FROM kunde
RIGHT JOIN (auftrag
JOIN auftragsposition ON AID = A_AID) ON KID = K_KID;
```

### 2. Aufgabe

```
SELECT KID, KundengrBez
FROM Kundengruppe FULL JOIN kunde ON (KG_KGID = KGID);
```

**Alternativbefehl:**

```
SELECT KID, KundengrBez
FROM Kundengruppe
LEFT JOIN kunde ON (KG_KGID = KGID)
UNION
SELECT KID, KundengrBez
FROM Kundengruppe
RIGHT JOIN kunde ON (KG_KGID = KGID);
```

## 4.7 SELF-JOIN

### 1. Aufgabe

```
SELECT m.Name AS Mitarbeiter, v.Name Vorgesetzter, Bezeichnung AS Abteilung
FROM mitarbeiter v
JOIN (mitarbeiter m
JOIN abteilung ON m.AB_ABID = ABID)
ON v.MID = m.M_Vorgesetzter
WHERE m.AB_ABID = v.AB_ABID;
```

### 2. Aufgabe

```
SELECT m.Name AS Mitarbeiter, AID, v.Name AS Vorgesetzter
FROM (mitarbeiter v
JOIN mitarbeiter m ON m.MID = v.M_Vorgesetzter)
JOIN auftrag ON M_MID = m.MID;
```

## 5.3 Vereinigung (kommutativ)

### 1. Aufgabe

```
SELECT Status, Kundengrbez, k .*
FROM kundengruppe
JOIN (kunde k
JOIN auftrag ON KID = K_KID)
ON KG_KGID = KGID
WHERE Status = 'offen'
UNION
SELECT Status, Kundengrbez, k .*
FROM kundengruppe
JOIN (kunde k
JOIN auftrag ON KID = K_KID)
ON KG_KGID = KGID
WHERE KundengrBez = 'C';
```

### 2. Aufgabe

```
SELECT ARID AS Artikel_ID, Name AS Lieferant, Farbe, Preis
FROM artikel
JOIN lieferant ON LID = L_LID
WHERE L_LID = 'L2'
UNION
SELECT ARID AS Artikel_ID, Name AS Lieferant, Farbe, Preis
FROM artikel
JOIN lieferant ON LID = L_LID
WHERE Farbe = 'schwarz'
UNION
SELECT ARID AS Artikel_ID, Name AS Lieferant, Farbe, Preis
FROM artikel
JOIN lieferant ON LID = L_LID
WHERE Preis > 40;
```

## 5.4 Durchschnitt (kommutativ)

### 1. Aufgabe

```
SELECT Positionsmenge, Einzelpreisrabatt
FROM auftragsposition
WHERE Positionsmenge > 3
INTERSECT
SELECT Positionsmenge, Einzelpreisrabatt
FROM auftragsposition
WHERE Einzelpreisrabatt >= 5
ORDER BY Positionsmenge ASC;
```

**Alternativbefehl für XAMPP:**

```
SELECT Positionsmenge, Einzelpreisrabatt
FROM auftragsposition ap
WHERE Positionsmenge > 3
AND Einzelpreisrabatt >= 5
ORDER BY Positionsmenge ASC;
```

### 2. Aufgabe

```
SELECT Name, Status, LieferDatum
FROM auftrag
JOIN mitarbeiter ON MID = M_MID
WHERE Name = 'Graf'
INTERSECT
SELECT Name, Status, LieferDatum
FROM auftrag
JOIN mitarbeiter ON MID = M_MID
WHERE Status = 'geliefert';
```

**Alternativbefehl für XAMPP:**

```
SELECT Name, Status, LieferDatum
FROM auftrag
JOIN mitarbeiter ON MID = M_MID
WHERE Name = 'Graf'
AND Status = 'geliefert';
```

## 5.5 Differenzmenge und Komplement (nicht kommutativ)

### 1. Aufgabe

```
SELECT Bezeichnung, Stadt
FROM abteilung
JOIN standort ON SID = S_SID
WHERE Stadt != 'Köln'
ECXEPT
SELECT Bereich, Stadt
FROM abteilung
JOIN standort ON SID = S_SID
WHERE Stadt = 'Berlin';
```

**Alternativbefehl für XAMPP:**

```
SELECT Bezeichnung, Stadt
FROM abteilung
JOIN standort ON SID = S_SID
WHERE Stadt != 'Köln'
AND NOT Stadt = 'Berlin';
```

### 2.Aufgabe

```
SELECT APID, Positionswert, Name
FROM kunde JOIN
(auftrag a JOIN auftragsposition ON AID = A_AID)
ON KID = K_KID
WHERE Anrede != 'Frau'
EXCEPT
SELECT APID, Positionswert, Name
FROM kunde JOIN (auftrag JOIN auftragsposition on AID = A_AID)
ON KID = K_KID
WHERE Positionswert < 150;
```

**Alternativbefehl für XAMPP:**

```
SELECT APID, Positionswert, Name
FROM kunde JOIN (auftrag
JOIN auftragsposition on AID = A_AID)
ON KID = K_KID
WHERE Anrede != 'Frau'
AND NOT Positionswert < 150;
```

## 5.6 Symmetrische Differenz (kommutativ)

### 1. Aufgabe

```
SELECT Status, Auftragsdatum
FROM auftrag
WHERE Auftragsdatum = '2012-10-05'
XOR STATUS = 'offen';
```

### 2. Aufgabe

```
SELECT Preis, Positionswert, Positionsmenge
FROM auftrag
JOIN (auftragsposition
JOIN artikel ON ARID = AR_ARID) ON AID = A_AID
WHERE (Positionsmenge > 3
AND Positionswert > 400)
XOR Preis > 150
GROUP BY Preis;
```

## 6.1 SUM()

### 1. Aufgabe

```
SELECT AR_ARID, SUM(Positionsmenge) AS Artikelmenge
FROM auftragsposition
GROUP BY AR_ARID
ORDER BY Artikelmenge ASC;
```

### 2. Aufgabe

```
SELECT KID, SUM(Positionswert) AS Summe_pro_Kunde
FROM auftragsposition
JOIN (auftrag
JOIN kunde ON KID = K_KID) ON A_AID = AID
GROUP BY KID;
```

## 6.2 COUNT()

### 1. Aufgabe

```
SELECT Artikelbez AS Artikelname, COUNT(KID) AS Anzahl_Kunde
FROM auftrag
JOIN (auftragsposition
JOIN artikel ON ARID = AR_ARID) ON AID = A_AID
JOIN kunde ON KID = K_KID
WHERE Farbe = 'schwarz'
GROUP BY Artikelname;
```

### 2. Aufgabe

```
SELECT A_AID, COUNT(Positionsmenge) AS Anzahl
FROM auftragsposition
GROUP BY A_AID
HAVING Anzahl > 1
ORDER BY A_AID DESC;
```

### 3. Aufgabe

```
SELECT KID, COUNT(*) AS Anzahl_der_Auftragspositionen
FROM auftragsposition
JOIN (auftrag
JOIN kunde ON KID = K_KID) ON AID = A_AID
GROUP BY KID
ORDER BY Anzahl_der_Auftragspositionen ASC;
```

### 4. Aufgabe

```
SELECT COUNT(DISTINCT Einzelpreisrabatt) AS Anzahl_ Einzelpreisrabatte
FROM auftragsposition;
```

## 6.3 AVG()

### 1. Aufgabe

```
SELECT AVG(Positionswert) AS Kunde_Schneider
FROM auftragsposition
JOIN (auftrag
JOIN kunde ON KID = K_KID) ON A_AID = AID
WHERE Name = 'Schneider';
```

### 2. Aufgabe

```
SELECT A_AID, AVG(Einzelpreisrabatt) AS Rabatt
FROM auftragsposition
GROUP BY A_AID
HAVING Rabatt > 5
ORDER BY Rabatt DESC;
```

## 6.4 MAX() UND MIN()

### 1. Aufgabe

```
SELECT K_KID, MIN(AuftragsDatum) AS Erster_Auftrag
FROM auftrag
GROUP BY K_KID
ORDER BY K_KID ASC;
```

### 2. Aufgabe

```
SELECT M_MID, MAX(AuftragsDatum) AS letzter_Auftrag
FROM auftrag
GROUP BY M_MID
ORDER BY M_MID ASC;
```

## 3. Aufgabe

```
SELECT A_AID, MAX(Positionsmenge) , MIN(Positionsmenge)
FROM auftragsposition
GROUP BY A_AID
ORDER BY A_AID DESC;
```

## 4. Aufgabe

```
SELECT MID, MAX(Positionswert)
FROM auftragsposition
JOIN (auftrag
JOIN mitarbeiter ON M_MID = MID) ON A_AID = AID
GROUP BY MID
ORDER BY MID ASC;
```

# Alter ausrechnen:

Die Zeitrechnung in phpMyAdmin beginnt mit dem Jahr 0!

**1.) SELECT DATEDIFF(CURDATE(), Geburtstag) DIV 365.25 AS Alter**

| Syntaxbefehl | Erklärung | Beispiel |
|---|---|---|
| `CURDATE()` | gibt das heutige Datum aus. | 2013-05-04 |
| `DATEDIFF` | subtrahiert den Geburtstag vom heutigen Datum, als Ergebnis erscheint das Alter als Tage. | 2013-05-04 – 1992-01-20 = 7775 |
| `DIV 365` `(Ganzzahldivision)` | dividiert die Anzahl der Tage durch 365, um das Alter rauszubekommen. | 7775 DIV 365 = 21 |

**2.) SELECT (TO_DAYS(CURDATE())-TO_DAYS(Geburtstag)) DIV 365.25 AS Alter**

| Syntaxbefehl | Erklärung | Beispiel |
|---|---|---|
| `CURDATE()` | gibt das heutige Datum aus. | 2013-05-04 |
| `TO_DAYS(CURDATE())` | gibt das heutige Datum in Tagen aus. | 2013-05-04 = 735357 |
| `TO_DAYS(Geburtstag)` | gibt den Geburtstag in Tagen aus. | 1992-01-20 = 727582 |
| `DIV 365` `(Ganzzahldivision)` | dividiert die Anzahl der Tage durch 365, um das Alter rauszubekommen. | (735357 – 727582) DIV 365 = 21 |

**3.) SELECT (YEAR(CURRENT_DATE)–YEAR(Geburtstag)) – (RIGHT(CURRENT_DATE, 5) < RIGHT(Geburtstag, 5)) AS Alter**

| Syntaxbefehl | Erklärung | Beispiel |
|---|---|---|
| **CURRENT_DATE** | gibt das heutige Datum aus. | 2013-05-04 |
| **YEAR(CURRENT_DATE)** | gibt das aktuelle Jahr aus. | 2013 |
| **YEAR(Geburtstag)** | gibt das Geburtsjahr aus. | 1992 |
| **RIGHT(CURRENT_DATE, 5)** | vom aktuellen Datum die letzten 5 Stellen von rechts auswählen, um den Monat und den Tag herauszufiltern. | 05-04 |
| **RIGHT( Geburtstag, 5)** | vom Geburtstag die letzten 5 Stellen von rechts auswählen, um den Geburtsmonat und –tag herauszufiltern. | 01-20 |
| **RIGHT(CURRENT_DATE, 5) < RIGHT(Geburtstag, 5)** | Vergleicht den aktuellen Monat und den Tag mit dem Geburtsmonat und –tag, und gibt aus, ob die Person dieses Jahr bereits Geburtstag hatte oder nicht. | Wenn `RIGHT(CURRENT_DATE, 5)` kleiner ist als `RIGHT(Geburtstag, 5)` hatte die Person in diesem Jahr noch keinen Geburtstag!<br>(Logischer Wert des Ausdrucks 1, also 1 vom Unterschied vom aktuellen Jahr und Geburtsjahr abziehen.) |

# Zeitberechnung:

```
SELECT TIMEDIFF(Ende, Beginn)
```

| Syntaxbefehl | Erklärung | Beispiel |
|---|---|---|
| `TIMEDIFF(Ende, Beginn)` | gibt den Zeitraum zwischen der Startzeit und der Endzeit in Minuten zurück. | 45 |

# STUDENT

## Aufgabe 1

```
SELECT Vorname, Name, DATEDIFF(CURDATE(), Geburtstag) DIV 365.25 AS
Studentenalter
FROM student
WHERE Geburtstag = (SELECT MAX(Geburtstag)
                    FROM student WHERE Anrede ='Frau')
AND Anrede ='Frau'
UNION
SELECT Vorname, Name, DATEDIFF(CURDATE(), Geburtstag) DIV 365.25 AS
Studentenalter
FROM student
WHERE Geburtstag = (SELECT MAX(Geburtstag)
                    FROM student WHERE Anrede ='Herr')
AND Anrede ='Herr';
```

## Aufgabe 2

```
CREATE VIEW new_table AS
SELECT Geburtstag, DATEDIFF(CURDATE(), Geburtstag) DIV 365.25 AS
Studentenalter FROM student; //Zwischentabelle erstellen

SELECT COUNT(DISTINCT Studentenalter) AS Anzahl_Versch_Alter,
COUNT(DISTINCT Geburtstag) AS Anzahl_Versch_Geburtstage FROM new_table
ORDER BY Anzahl_Versch_Geburtstage;
```

## Aufgabe 3

```
SELECT Vorname, Name, Studiengang, DATEDIFF(CURDATE(), Geburtstag) DIV
365.25 AS Studentenalter, Fachbezeichnung AS Übung
FROM student
JOIN (vorlesungsteilnahme v
JOIN (vorlesung JOIN übung u ON VID = u.V_VID) ON v.V_VID = VID)
ON SID = S_SID
WHERE DATEDIFF(CURDATE(),Geburtstag) DIV 365 = 21
ORDER BY Name;
```

## Aufgabe 4

```
SELECT Fachbezeichnung, COUNT(TID) AS Anzahl_Teilnehmer
FROM student
JOIN (vorlesungsteilnahme
JOIN vorlesung ON VID = V_VID) ON SID = S_SID
WHERE Studiengang = 'Informatik'
AND Fachbezeichnung = 'Statistik';
```

## Aufgabe 5

```
SELECT SID, Semester, TutorFür, COUNT(TID) AS Anzahl_der_Vorlesungen
FROM student JOIN vorlesungsteilnahme ON SID = S_SID
WHERE semester = 4
XOR TutorFür IS NOT NULL
GROUP BY SID;
```

## Aufgabe 6

```
SELECT SID AS Tutor, Fachbezeichnung
FROM student
JOIN (vorlesung
JOIN vorlesungsteilnahme ON V_VID = VID) ON SID = S_SID
WHERE TutorFür IS NOT NULL
ORDER BY SID ASC;
```

## Aufgabe 7 a

```
SELECT SID
FROM student
JOIN (vorlesungsteilnahme
JOIN (vorlesung
JOIN professor d ON PID = P_PID) ON V_VID = VID) ON SID = S_SID
WHERE d.Name = 'Gerstner'
GROUP BY SID
HAVING COUNT(*) = 1
ORDER BY SID DESC;
```

## Aufgabe 7 b

```
SELECT SID, Fachbezeichnung, TIMEDIFF(Ende, Beginn) AS Vorlesungsdauer
FROM student
JOIN (vorlesungsteilnahme
JOIN (vorlesung
JOIN professor p ON PID = P_PID) ON V_VID = VID) ON SID = S_SID
WHERE p.Name = 'Gerstner'
GROUP BY SID, Fachbezeichnung
ORDER BY SID ASC;
```

## Aufgabe 8

```
SELECT SID, Matrikelnummer, Fachbezeichnung
FROM student
JOIN (vorlesungsteilnahme
JOIN vorlesung ON VID = V_VID) ON SID = S_SID
WHERE Studiengang = 'Wirtschaftsinformatik'
INTERSECT
SELECT SID, Matrikelnummer, Fachbezeichnung
FROM student
JOIN (vorlesungsteilnahme
JOIN vorlesung ON VID = V_VID) ON SID = S_SID
WHERE Fachbezeichnung = 'Datenbanken'
ORDER BY Matrikelnummer;
```

### Alternativbefehl für XAMPP:

```
SELECT SID, Matrikelnummer, Fachbezeichnung
FROM student
JOIN (vorlesungsteilnahme
JOIN Vorlesung ON VID = V_VID) ON SID = S_SID
WHERE Studiengang = 'Wirtschaftsinformatik'
AND Fachbezeichnung = 'Datenbanken'
ORDER BY Matrikelnummer;
```

## Aufgabe 9

```
SELECT SID, ROUND(AVG(Note)) AS Durchschnitts_Note
FROM student JOIN prüfungsanmeldung ON SID = S_SID
GROUP BY SID
HAVING Durchschnitts_Note < 3;
```

## Aufgabe 10

```
SELECT SID, PRAID
FROM prüfungsanmeldung RIGHT JOIN student
ON S_SID = SID
WHERE PRAID IS NULL
ORDER BY SID ASC;
```

### Alternativbefehl:

```
SELECT SID
FROM student
WHERE SID NOT
IN (SELECT S_SID
    FROM prüfungsanmeldung)
ORDER BY SID ASC;
```

## Aufgabe 11

```
SELECT SID
FROM student JOIN (prüfungsanmeldung
JOIN prüfung ON PRID = PR_PRID) ON SID = S_SID
WHERE ErstVersuch = 5
GROUP BY SID
ORDER BY SID ASC;
```

### Alternativbefehl:

```
SELECT DISTINCT SID
FROM student JOIN (prüfungsanmeldung
JOIN prüfung ON PRID = PR_PRID) ON SID = S_SID
WHERE ErstVersuch = 5
ORDER BY SID ASC;
```

## Aufgabe 12 a

```
SELECT SID, Fach, Status , Note AS mit_Erfolg
FROM student s
JOIN (prüfungsanmeldung
JOIN prüfung ON PR_PRID = PRID) ON SID = S_SID
WHERE Note < 5
ORDER BY Note ASC;
```

## Aufgabe 12 b

```
SELECT SID, Fach, Status , Note AS mit_Erfolg
FROM student s
JOIN (prüfungsanmeldung
JOIN prüfung ON PR_PRID = PRID) ON SID = S_SID
WHERE Note < 5
MINUS
SELECT SID, Fach, Status , Note AS mit_Erfolg
FROM student s
JOIN (prüfungsanmeldung
JOIN prüfung ON PR_PRID = PRID) ON SID = S_SID
AND ErstVersuch IS NULL
ORDER BY Note ASC;
```

### Alternativbefehl für XAMPP:

```
SELECT SID, Fach, Status, Note AS Bestanden
FROM student s
JOIN (prüfungsanmeldung
JOIN prüfung ON PR_PRID = PRID) ON SID = S_SID
WHERE Note <5
AND ErstVersuch IS NULL
ORDER BY Note ASC;
```

## Aufgabe 13

```
SELECT SID, Note, Wohnort
FROM prüfung
JOIN (prüfungsanmeldung
JOIN student ON SID = S_SID) ON PRID = PR_PRID
WHERE Wohnort = 'Nürnberg'
MINUS
SELECT SID, Note, Wohnort
FROM prüfung
JOIN (prüfungsanmeldung
JOIN student ON SID = S_SID) ON PRID = PR_PRID
WHERE Note = 1
OR Note = 5;
```

**Alternativbefehl:**

```
SELECT SID, Note, Wohnort
FROM prüfung
JOIN (prüfungsanmeldung
JOIN student ON SID = S_SID) ON PRID = PR_PRID
WHERE Wohnort = 'Nürnberg'
AND NOT Note != 1
OR Note != 5;
```

## Aufgabe 14

```
SELECT SID, Note, Fach, p.Name AS Professor
FROM student
JOIN (Prüfungsanmeldung JOIN (prüfung LEFT JOIN professor p ON PID = P_PID)
ON PR_PRID = PRID) ON SID = S_SID
WHERE Note IS NOT NULL
AND Note != 5
ORDER BY Fach ASC;
```

## Aufgabe 15 a

```
SELECT SID, P_PID, Fachbezeichnung
FROM student
LEFT JOIN (vorlesungsteilnahme
JOIN vorlesung ON V_VID = VID) ON SID = S_SID
ORDER BY Fachbezeichnung ASC;
```

## Aufgabe 15 b

```
SELECT Name, Fachbezeichnung, COUNT(TID) AS aktuelle_Teilnehmerzahl
FROM vorlesungsteilnahme
JOIN (vorlesung
JOIN professor p ON PID = P_PID) ON V_VID = VID
GROUP BY Fachbezeichnung, Name
ORDER BY COUNT(TID) DESC;
```

## Aufgabe 16

```
SELECT Fachbezeichnung, Name
FROM professor LEFT JOIN vorlesung ON P_PID = PID
ORDER BY Fachbezeichnung DESC;
```

### Alternativbefehl:

```
SELECT Fachbezeichnung, Name
FROM professor
JOIN vorlesung ON P_PID = PID
UNION
SELECT " ", Name
FROM professor
WHERE PID NOT
IN (SELECT P_PID FROM vorlesung)
ORDER BY Fachbezeichnung DESC;
```

## Aufgabe 17 a

```
SELECT COUNT(UID) AS Anzahl_der_Übungen
FROM übung u JOIN vorlesung ON VID = V_VID
WHERE TIMEDIFF(u.Ende, u.Beginn) >= "00:45:00"
ORDER BY Dauer ASC;
```

## Aufgabe 17 b

```
SELECT TIMEDIFF(u.Ende, u.Beginn) AS Dauer, Fachbezeichnung AS Übung
FROM übung u JOIN vorlesung ON VID = V_VID
GROUP BY Fachbezeichnung, Dauer
HAVING Dauer >= "00:45:00"
ORDER BY Dauer ASC;
```

## Aufgabe 17 c

```
SELECT TIMEDIFF(Ende, Beginn) AS Vorlesungsdauer, COUNT(*) AS
Anzahl_der_Vorlesungen
FROM vorlesung
GROUP BY Vorlesungsdauer
HAVING Vorlesungsdauer IS NOT NULL
ORDER BY Vorlesungsdauer DESC;
```

### Aufgabe 18

```
SELECT Beginn, Ende, TIMEDIFF( Ende, Beginn ) AS Dauer, v.RaumNr, Name AS
Professor, Fachbezeichnung, Tag
FROM vorlesung v
LEFT JOIN professor ON PID = P_PID
WHERE v.RaumNr IS NOT NULL
AND Beginn = "08:00:00";
```

## PROFESSOR

### Aufgabe 19

```
CREATE VIEW Professor_Student AS
SELECT PID, SID, COUNT(SID) AS Anzahl
FROM student JOIN (vorlesungsteilnahme JOIN (vorlesung JOIN professor ON
PID = P_PID) ON VID = V_VID) ON SID = S_SID
GROUP BY SID, PID;

SELECT PID, SID
FROM professor_student
GROUP BY PID, SID
HAVING COUNT(*)= SUM(Anzahl)
ORDER BY PID;
```

## PRÜFUNG

### Aufgabe 20

```
SELECT Fach, ROUND(AVG(Note),2) AS Durchschnittsnote , MIN(Note) AS
Beste_Note, Name
FROM professor
JOIN (prüfung JOIN prüfungsanmeldung ON PR_PRID = PRID) ON PID = P_PID
WHERE Note < 5.0
GROUP BY Fach, Name
HAVING Durchschnittsnote <= 3.5
ORDER BY Durchschnittsnote ASC;
```

## Internet


- Michael Olschimke(2010): Conformance-Anweisungen
  http://de.dwhwiki.info/konzepte/sql
  (letzter Zugriff: 13. Januar 2013)

- Fabian Simon (2012): Referentielle Integrität
  http://www.wirtschaftsinformatik-24.de/datenbanken/referentielle-integritaet.php
  (letzter Zugriff: 08. Februar 2013)

- Lutz Hunger (2012): Aggregatfunktionen
  http://www.teialehrbuch.de/Kostenlose-Kurse/SQL/14750-Aggregatfunktionen.html
  (letzter Zugriff: 12. Dezember 2012)

- Robert Warnke (2012): Gruppierungen
  http://rowa.giso.de/oracle/latex/Komplexere_SQL_Abfragen.html#SECTION00234000000000000000
  (letzter Zugriff: 03. Dezember 2012)

- Wikimedia Foundation Inc. (2011): Einführung in SQL: WHERE-Klausel im Detail
  http://de.wikibooks.org/wiki/Einf%C3%BChrung_in_SQL:_WHERE-Klausel_im_Detail
  (letzter Zugriff: 28. November 2012)

- Rolf Walter (2009): Grundbegriffe der Mengenlehre
  http://www.mathematik.uni-dortmund.de/lsvii/Preprints/mengen.pdf
  (letzer Zugriff: 28. November 2012)

- Alex Pratzer (2012): MySQL-Anweisungen lernen - phpMyAdmin nutzen
  http://www.php-kurs.com/mysql-anweisungen-lernen---phpmyadmin-nutzen.htm
  (letzter Zugriff: 12. Dezember 2012)

- SRS Solutions (2010): Einrichten eines eigenen Webservers mit XAMPP
  http://www.hightech-journal.net/einrichten-eines-eigenen-webservers-mit-xampp
  (letzter Zugriff: 30.Januar 2013)

- Alexander Regantisch(2005-2011): CMS-Vorstellung: PHPWCMS
  http://www.mastblau.com/2005-11-21/cms-vorstellung-phpwcms/
  (letzter Zugriff: 11. Februar 2013)

## Bücher


- G. Taylor, Allen (2011): **SQL für DUMMIES**, 5. Auflage, Weinheim

- Preiß, Nikolai (2007): **Entwurf und Verarbeitung relationaler Datenbanken, Eine durchgängige und praxisorientierte Vorgehensweise**, Stuttgart