SQL

– Structured Query Language

mit

Inhalt

1 Vorwort

Wir danken unserem geschätzten Kollegen Winfried Bartsch, der die Arbeitsgrundlage für das vorliegende Skript "Datenbanken – MySQL" geliefert hat.

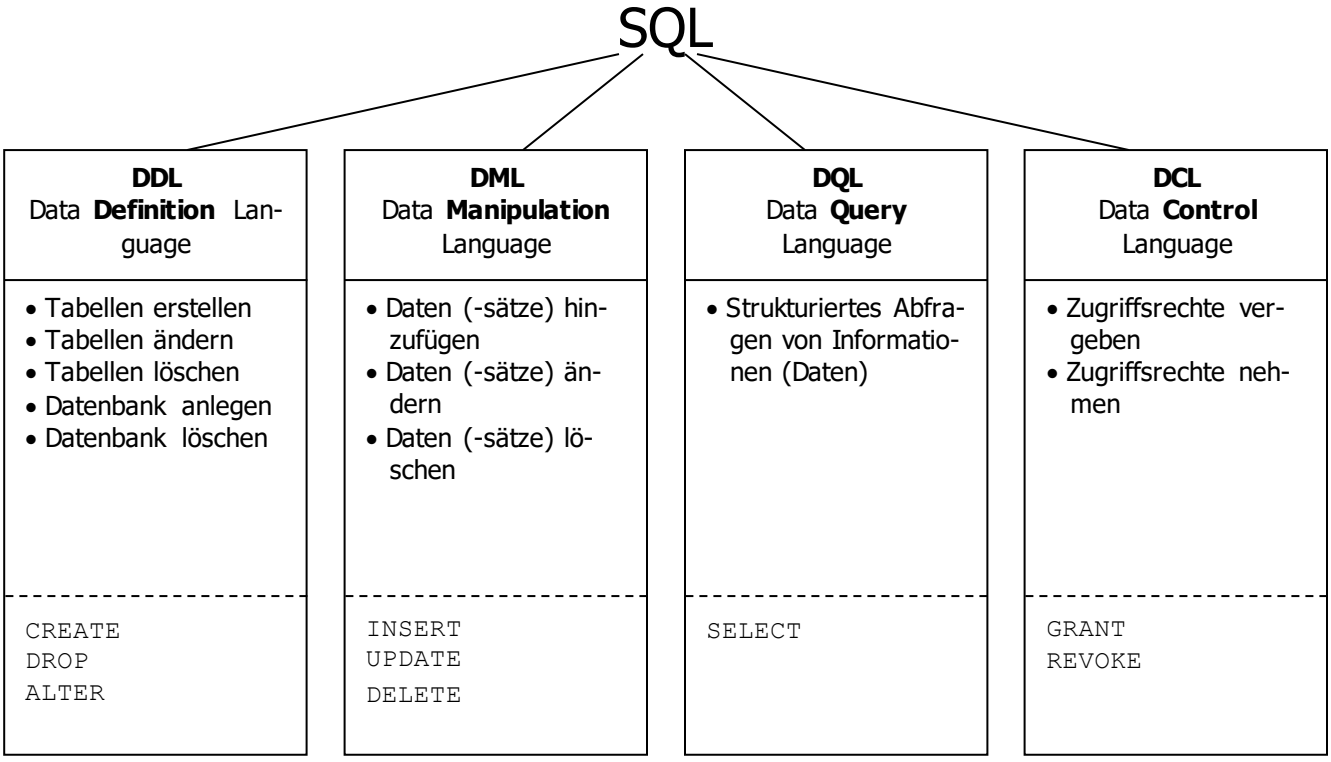
Die Fachschaft Anwendungsentwicklung und Programmierung der städtischen Berufsschule für Informationstechnik München.

2 Einführung in SQL

SQL - Structured Query Language ist die de-facto-Standardsprache, mit der sich Daten aus relationalen Datenbanken abrufen und manipulieren lassen. Der Name SQL legt nahe, dass es sich hier lediglich um eine Abfragesprache handelt. SQL wurde jedoch von Anfang an als vollständige Datenbanksprache entwickelt, mit der alle Bereiche einer Datenbankanwendung abgedeckt werden sollten:

- Datendefinition dient zur Definition von Datenstrukturen innerhalb der Datenbank
- Informationen aktualisieren (einfügen, ändern, löschen)
- Informationen abfragen
- Datenschutz dient zur Definition von Zugriffsrechten auf Objekte in der Datenbank
- Datensicherheit dient zur Gewährleistung der Ablaufintegrität
- Benutzerberechtigungen für den Gebrauch und Zugriff verwalten
- Datenorganisation dient zur Verwaltung der gesamten Datenbank

2.1 Teilsprachen von SQL



3 Entwicklung

SQL entstand Ende der 70er Jahre bei IBM (San Jose, Kalifornien). Die Entwicklung war ursprünglich für das IBM-Produkt DB2 vorgesehen (ein RDBMS = Relationales Datenbank-Managementsystem). Die Entwicklung kann an folgenden Eckpunkten festgemacht werden:

- Relationales Datenbankmodell (Edgar G. Codd, Entwickler bei IBM, 1970)
- Erste Datenbanksprache: SEQUEL (Structured English Query Language, 1974)
- SQL1 oder SQL-86 (erstmals Normierung durch ANSI)
- SQL2 oder SQL-92
- SQL3 oder SQL-99
- SQL2006 (Verwendung von XML)
- SQL2008 (aktuelle Revision des SQL-Standards)
- SQL – MySQL

Wir werden das Datenbankmanagementsystem MySQL einsetzen, welches auf dem Standard von SQL beruht und darüber hinaus eigene Features anbietet. Wir benutzen MySQL als Einstieg in das Erlernen eines DBMS. Wer sich damit zurechtfindet, wird leicht(er) mit anderen relationalen Systemen arbeiten können. Zudem ist MySQL mittlerweile die populärste Open-Source-Datenbank der Welt.

4 DDL – Data Definition Language

SQL als Definitionssprache
Zunächst müssen die im Entwurf des relationalen Modells festgelegten Tabellen "hergestellt" werden. Dazu dient die Datendefinitionssprache DDL (steht für Data Definition Language). Sie umfasst alle Klauseln, die mit der Definition von Tabellen, Typen, Wertebereichen und Integritätsbedingungen zu tun haben. Im Vordergrund steht dabei die Definition von Relationenschemata (das sind die Tabellen).
Eine relationale DDL sollte (nach Codd) mindestens folgende Bestandteile definieren:

- Relationenschemata
- Attribute
- Wertebereiche
- Primärschlüssel und Fremdschlüssel

Benutzt man SQL zur Definition der Datenbank-Anwendung auf der externen, konzeptuellen oder internen Ebene, so können folgende DDL-Befehle zugeordnet werden:

Einrichten der Datenbank	`create database`
Externe Ebene	`create view` `drop view`

Konzeptionelle Ebene	`create table`
	`alter table`
	`drop table`

Interne Ebene	`create index`
	`alter index`
	`drop index`

Das Beispiel "Schule"
Im Weiteren wird die Syntax der SQL-Sprache am Beispiel der Datenbank Schule erklärt. Dazu liegen zunächst zwei Tabellen vor:

Auszubildender							
a_id	a_name	a_vorname	a_gebdat	a_gehalt	k_id	a_ort	a_plz
1	Schmitt	Johann	1975-12-31	560.00	4	Attenkirchen	85173
2	Huber	Max	1974-02-20	420.00	1	Zolling	86121
3	Wolf	Rita	1984-02-22	250.00	3	Garmisch-Partenkirchen	82180
4	Jonas	Sofie	1984-04-02	595.00	2	Attenkirchen	85173
5	Wolf	Sabrina	1975-01-01	380.00	4	München	80992
6	Holz	Stefan	1976-02-28	380.00	1	Augsburg	86153

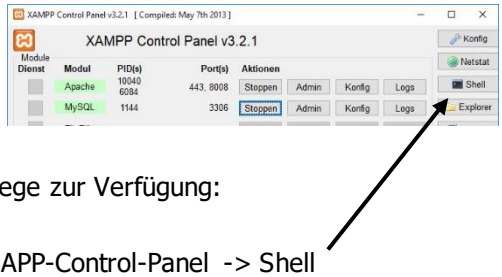
Klasse	
k_id	k_bez
1	Marketing
2	Einkauf
3	Programmierung
4	Zentrale

Anschließend an die Erklärung jedes SQL-Befehles finden Sie dann Aufgaben die sich auf die zu erstellende Datenbank "Foodshop" beziehen und welche Sie fortlaufend entwickeln werden!

4.1 Nutzung des MySQL DBMS an der BS-Informationstechnik

4.1.1 mit USB-Stick:

- Datenbanken: XAMPP starten (Control-Panel öffnet sich)
- SQL-Server starten (optional Apache starten)



Als Clients für den Zugriff auf das System stehen Ihnen mehrere Wege zur Verfügung:

- Shell: XMAPP-Control-Panel -> Shell
  am SQL-Server anmelden : `mysql –u root`
- MySQL Workbench (App auf Ihrem Stick): neue Connection (+), Titel: local
- PhpMyAdmin (Apache-Server nötig, s.o.): Browser: http://localhost:8008/phpmyadmin

4.1.2 ohne USB-Stick:

Falls sich Ihr System über den Stick nicht korrekt starten lässt gibt es folgende Alternative:
Kopieren Sie die Datei `"_mysql_server.zip"` aus dem Ordner `"[Pfad_KlassenLW\Vorlagen]\AP12"` auf ihren Desktop. Entpacken Sie die Datei anschließend in das Laufwerk `"H:\"` und folgen Sie den Anweisungen der Datei `"MySQL_readme.txt"`!

4.1.3 MySQL Benutzer und Passwort

Zur Anmeldung am DBMS nutzen Sie den Benutzer "root". Ein Passwort benötigen Sie nicht!

4.2 Anlegen einer Datenbank - CREATE DATABASE

Den Entwurf einer Datenbank bestimmen viele Faktoren:

- Sicherheit, Zugriffsrechte, Benutzerberechtigungen
- Verfügbarer Plattenplatz, Speicherplatzreservierung, -verwaltung, Allokation
- Geschwindigkeit beim Suchen und Abrufen der Datenbank, Suchstrategien
- Geschwindigkeit beim Aktualisieren der Datenbank, Zugriffsmechanismen
- Performanz beim Abruf von Daten aus verknüpften Tabellen
- Unterstützung von temporären Tabellen
- Tabellenstruktur (Normalisierung [viele Tabellen] gegenüber Performanz)

Diese Faktoren werden nun bei manchen DBMS automatisch festgelegt, bei anderen können die Parameter manuell (je nach angebotenen Optionen) definiert werden.
Die Syntax von CREATE DATABASE ist in den jeweiligen Implementierungen recht unterschiedlich. Auf jeden Fall muss zuerst eine Datenbank initialisiert werden:

Syntax:

```
CREATE DATABASE <Datenbankname>;
```

Beispiel:

```
CREATE DATABASE Schule;
```

4.3 Liste aller im DBMS angelegten Datenbanken

Syntax:

```
SHOW DATABASES;
```

Aufgabe 1:

- Erstellen Sie die Datenbank „FoodShop"!
- Lassen Sie sich alle Datenbanken Ihres Systems anzeigen und prüfen Sie, ob die Datenbank FoodShop tatsächlich angelegt wurde!

4.4 Erzeugen von Tabellen - CREATE TABLE

Wenn einmal eine Datenbank angelegt ist, können in ihr nun die entsprechenden Tabellen erstellt werden. Vorher muss die gewünschte Datenbank zur Nutzung aktiviert werden:

```
USE Schule;
```

Mit der CREATE TABLE-Anweisung wird eine neue Tabelle angelegt.
Syntax:

```
CREATE TABLE <Tabellenname> (
       <feld> <datentyp> [ NOT NULL ] [UNIQUE],
       ……
       PRIMARY KEY(<Spaltenname>)
);
```

4.4.1 NOT NULL

Mit NOT NULL wird die Eingabe eines Datenwertes für das entsprechende Datenfeld erzwungen. UNIQUE legt die Einmaligkeit des Wertes fest.

4.4.2 PRIMARY KEY

Mit PRIMARY KEY wird der Primärschlüssel der Tabelle festgelegt. Alle Felder des Primärschlüssels müssen mit NOT NULL gekennzeichnet werden.

4.4.3 AUTO_INCREMENT

Mit Hilfe der AUTO_INCREMENT Option können eindeutige Werte für mit INTEGER definierte Felder erzeugt werden. Mit jedem neuen Datensatz wird der Wert eines AUTO_INCREMENT Feldes standardmäßig um jeweils '1' erhöht. Dies ist insbesondere für Primärschlüsselfelder nützlich!
Die im Beispiel genutzten Datentypen werden im Abschnitt "4.4.4 Wichtige Datentypen" auf der nächsten Seite genauer erklärt!

Beispiele:

```
CREATE TABLE Auszubildender(
       a_id INT(3) AUTO_INCREMENT,
       a_name VARCHAR(30),
       a_vorname VARCHAR(30),
       a_gebdat DATE,
       a_gehalt DECIMAL(9,2),
       k_id INT(3),
       a_ort VARCHAR(30),
       a_plz CHAR(5),
       PRIMARY KEY(a_id)          #Primärschlüssel
);
```

```
CREATE TABLE Klasse(
       k_id INT(3) PRIMARY KEY,
       k_bez VARCHAR(35) NOT NULL
);
```

Aufgabe 2:

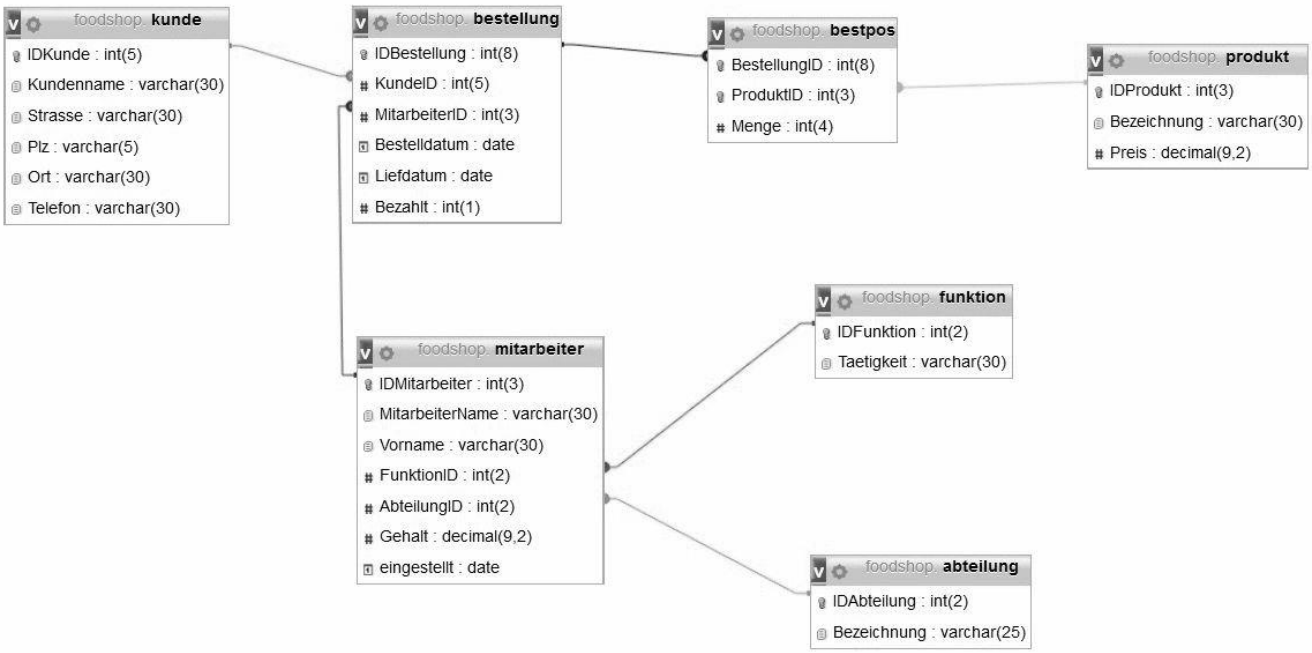
- Erstellen Sie nun für die Datenbank FoodShop entsprechend dem auf der folgenden Seite abgebildeten logischen Datenbankmodell die Tabellen „Mitarbeiter", „Funktion" und „Abteilung"!

  Hinweis: Die Beziehungen zwischen den Tabellen werden an dieser Stelle nur implizit über die Primär- und Fremdschlüsselwerte in den Tabellen abgebildet. Explizit angelegt werden können Sie über den in Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.** und **Fehler! Verweisquelle konnte nicht gefunden werden.** erklärten `Foreign Key Constraint!`
- Für schnelle Schüler: Erstellen Sie die weiteren Tabellen der DB FoodShop!
- Für weniger schnelle Schüler: Die fehlenden Tabellen können Sie über folgenden Befehl generieren lassen:

SHELL: `source [Pfad_KlassenLW\Vorlagen]\AP12\DB_FoodShop_Aufgabe2_tabellen_ergaenzen.sql`

Workbench: `Menü 'File' -> 'Open SQL-Script -> Execute (Blitz-Symbol)`

Datenbank „Foodshop":



4.4.4 Wichtige Datentypen

Numerische Datentypen

INT (4 Byte)	Ganzzahl (Standard)	
BIGINT (8 Byte)	große Ganzzahl	
DECIMAL(G,N)	Fließkommazahl	Länge G inklusive N Nachkommastellen

Datums- und Uhrzeit-Datentypen

DATE Format: YYYY-MM-DD	Datum	1.1.1000 bis 31.12.9999
DATETIME Format: YYYY-MM-DD hh:mm:ss	Datum und Uhrzeit	1.1.1000, 0:00:00 Uhr bis 31.12.9999, 23:59:59
TIMESTAMP	Unixzeit	1.1.1970 bis 31.12.2036
TIME	Uhrzeit	Format hh:mm:ss
YEAR	Jahreszahl	1901 bis 2155

Zeichenketten-Datentypen

CHAR(G)	Zeichenkette mit fester Größe G	max. 255 Zeichen
VARCHAR(G)	Zeichenkette mit variabler Größe G	max. 255 Zeichen
BLOB	Binary large object (z.B. um Grafiken zu speichern)	max. 2^{16} Zeichen
TEXT	Zeichenketten > 255 Zeichen	max. 2^{16} Zeichen

4.5 Struktur einer Tabelle anzeigen - DESCRIBE und SHOW

Unter MySQL lässt sich die Struktur einer Tabelle, das relationale Design, anzeigen:

```
DESCRIBE Auszubildender; verkürzt geht auch: DESC Auszubildender;
```

`CREATE TABLE` Kommando einer Tabelle anzeigen

Unter MySQL lässt sich das create table Kommando für eine Tabelle anzeigen:

```
show create table Auszubildender;
```

Aufgabe 3:

- Lassen Sie sich die Tabellenstruktur aller selbst angelegten Tabellen noch einmal anzeigen und überprüfen Sie sorgfältig, ob die Struktur dem vorliegenden Tabellenmodell entspricht. Achten Sie auf mögliche Tippfehler, vergessene Attribute und fehlende Primärschlüssel!

4.6 Ändern einer Tabelle - ALTER TABLE

Mit ALTER TABLE kann die Struktur einer Tabelle nachträglich verändert werden. So können beispielsweise Spalten hinzufügt oder entfernt, Indizes erstellt oder gelöscht, der Typ einer vorhandenen Spalte geändert oder Spalten oder die Tabelle umbenannt werden.

Syntax:

```
ALTER TABLE <Tabellenname>
<alter_specification>;
```

4.6.1 Spalten hinzufügen - ADD

Um eine Spalte an eine bestimmte Position in einer Tabellenzeile hinzuzufügen, verwenden Sie FIRST oder AFTER <Spaltenname>. Standardmäßig werden neue Spalten am Ende eingefügt.

Beispiel:

```
ALTER TABLE Auszubildender
ADD a_mailadresse VARCHAR(30) AFTER a_gebdat;
```

4.6.2 Spaltenname, Definition, Typ ändern - CHANGE, MODIFY

Eine Spalte kann mit CHANGE <Spaltenname_alt> <spaltendefinition_neu> umbenannt werden. Hierbei ist die komplette Definition für die neue Spaltebezeichnung (inkl. Typ) nötig.

```
ALTER TABLE Auszubildender
CHANGE a_mailadresse a_email varchar(10);
```

Wenn der Typ bzw. die Länge einer Spalte, nicht aber deren Namen geändert werden soll, erfordert die CHANGE-Syntax trotzdem den alten und einen neuen Spaltennamen, auch wenn diese sich nicht unterscheiden.

```
ALTER TABLE Auszubildender
CHANGE a_email a_email VARCHAR(40);
```

Weitere Änderungsmöglichkeiten:

```
ALTER TABLE Auszubildender
DROP PRIMARY KEY;
```

MODIFY ist eine Alternative, um den Spaltentyp ohne Umbenennung der Spalte zu ändern.

```
ALTER TABLE Klasse
MODIFY k_id INTEGER AUTO_INCREMENT PRIMARY KEY;
```

ADD FOREIGN KEY …. REFERENCES – Referentielle Integrität herstellen – siehe 5.4.1

```
ALTER TABLE Auszubildender
ADD FOREIGN KEY (k_id) REFERENCES Klasse(k_id);
```

4.6.3 Spalte löschen – DROP

Eine Spalte wird mit DROP gelöscht.

```
ALTER TABLE <tabelle>
DROP <Spalte>;
```

4.6.4 Löschen einer Tabelle - DROP TABLE

Mit DROP TABLE stellt SQL einen Befehl zum vollständigen Löschen einer Tabelle aus einer Datenbank bereit. Die Tabelle wird zusammen mit allen zugehörigen Sichten und Indizes gelöscht. Die Ausführung des Befehls ist unwiderruflich.

```
DROP TABLE <tabelle>;
```

Aufgabe 4: ALTER TABLE

- Fügen Sie zur Tabelle „Mitarbeiter" die Spalte Geburtsdatum (Date) nach der Spalte Vorname hinzu!
- Ändern Sie die Spalte IDProdukt der Tabelle Produkt in einen Autowert.
  Siehe Abschnitt 4.4.3
- Löschen Sie die Spalte Geburtsdatum aus der Tabelle Mitarbeiter!

5 DML – Data Manipulation Language

MySQL als Manipulationssprache

Die Data Manipulation Language (DML) umfasst alle Anweisungen zur Beeinflussung der gespeicherten Daten. Grundlegende Operationen sind dabei: Eingabe, Änderung und Löschen von Daten. Die einzelnen Anweisungen werden mit den zuvor erzeugten Tabellen Auszubildender und Klasse der Datenbank Betrieb durchgespielt.

5.1 Einfügen - INSERT-Anweisung

Mit der INSERT-Anweisung werden Daten in Tabellen eingetragen. Dabei können sowohl ganze Zeilen als auch nur Teile von Tupeln (Datensätzen) mit Werten belegt werden. Die übrigen, nicht belegten Spalten bekommen jeweils den Standardwert NULL.

Die Daten können einmal durch eine SELECT-Anweisung aus einer anderen Tabelle ermittelt werden. Zum Zweiten kann ein einzelner Satz durch Angabe der Felder und deren Werte eingefügt werden. Die Datentypen der Werte in der Werteliste müssen mit denjenigen der zugehörigen Spalte übereinstimmen. Die SELECT-Anweisung wird im nächsten Kapital genauer beschrieben.

Syntax:

```
INSERT [INTO] <Tabellenname> <SELECT-Anweisung>;
oder
INSERT INTO <Tabellenname> (<Spalten>) VALUES(<Werte>);
```

Hinweis: Das Wort INTO ist dabei optional und kann auch entfallen.

Beispiele:

```
INSERT INTO Auszubildender
VALUES (1, 'Schmitt', 'Johann', '1975-12-17', 560.00, 4);
```

Genauso gut könnte man schreiben:

```
INSERT INTO Auszubildender (a_id, a_name, a_vorname, a_gebdat, a_gehalt, k_id)
VALUES (1, 'Schmitt', 'Johann', '1975-12-17', 560.00, 4);
```

Teile von Datensätzen werden so eingegeben:

```
INSERT INTO Auszubildender (a_id, a_name, a_vorname)
VALUES (5, 'Doll', 'Jan');
```

Mehrere Datensätze mit einem einzigen INSERT Kommando geht auch:

```
INSERT INTO Auszubildender (a_id, a_name, a_vorname)
VALUES (5, 'Doll', 'Jan'),
       (6, 'Juergens','Thomas'),
       (7, 'Kustermann','Beate');
```

Aus einer Tabelle können ebenfalls Daten eingespielt werden. Nach einer Fusion mit einem anderen Unternehmen A sollen jene Auszubildender in die Datenbank eingespielt werden. Dazu benutzen wir Informationen der Tabelle Auszubildender_A aus der Datenbank von Unternehmen A.

Auszubildender_A				
aa_id	aa_name	aa_vorname	aa_gebdat	aa_gehalt
1	Huber	Josef	1950-01-12	NULL
2	Huber	Lise	1956-10-13	NULL
3	Codd	Frank	1953-03-31	NULL
4	Date	Ursula	1974-10-18	NULL

Stimmen die beiden Tabellen in ihrer Struktur überein, geht folgendes Statement:

```
INSERT INTO Auszubildender
SELECT *
FROM Auszubildender A;
```

Wenn die Struktur der beiden Tabellen nicht übereinstimmt, das ist in unserem Fall mit den Tabellen Auszubildender und Auszubildender_A so, müssen die Spalten ausdrücklich angegeben werden:

```
INSERT INTO Auszubildender(a_id, a_name, a_vorname, a_gebdat)
SELECT aa_id, aa_name, aa_vorname, aa_gebdat
FROM Auszubildender_A;
```

Hinweis: Auch die Datentypen der Attribute von Quell- und Zieltabelle müssen übereinstimmen!

Aufgabe 5:

- Fügen Sie die Abteilungsbezeichnungen (Einkauf, Küche, Telefondienst, Produktion, Hausmeister) zusammen mit ihrem Primärschlüssel (durchnummeriert beginnend mit '1') in die Tabelle Abteilung ein!
- Fügen Sie anschließend alle übrigen Daten mit Hilfe des SQL-Skripts "DB_FoodShop_Aufgabe5_Daten_einfuegen.sql" ein! Nutzen Sie wiederum den `source` Befehl der bereits in Aufgabe 2: einmal angewendet wurde!

5.2 Ändern - UPDATE-Anweisung

Mit der UPDATE-Anweisung können vorhandene Daten in Tabellen modifiziert werden. In den ausgewählten Zeilen werden die Werte neu berechnet bzw. zugewiesen und eingetragen.
Syntax:

```
UPDATE <Tabellenname>
SET <Spalte> = <WERT> [,<Spalte> = <WERT>]
[WHERE <Bedingung> ];
```

Beispiele:

```
UPDATE Auszubildender
SET a_name = 'Lupus'
WHERE a_id = 10;
```

```
UPDATE Auszubildender
SET a_name = 'Lupus', a_gehalt = 400.00
WHERE a_id = 10;
```

5.3 Löschen - DELETE-Anweisung

Mit DELETE FROM werden Datensätze aus einer Datenbanktabelle gelöscht. Die Bedingung gibt an, um welche Datensätze es sich handelt.

```
DELETE FROM Auszubildender
WHERE a_id = 10;
```

Fehlt diese Bedingung, werden alle Zeilen einer Tabelle

```
DELETE FROM Auszubildender;
```

Aufgabe 6:

- Die Abteilungsbezeichnung „Hausmeister" ist nicht mehr zeitgemäß und soll in „Facility Management" umbenannt werden. Die Tabelle Abteilung muss daher geändert werden!
- Es gibt Überlegungen, den Telefondienst durch die übrigen Abteilungen selbst erledigen zu lassen. Welche Anweisung wäre nötig, um die Abteilung „Telefondienst" zu löschen? Schreiben Sie den passenden SQL-Befehl nachfolgend auf – führen Sie ihn NICHT AUS!

Befehl: ……..

5.4 Konsistenz und Integrität von Datenbanken

Ziel jeden Datenbankdesigns und späteren Betriebes von Datenbanken ist die möglichst vollständige Konsistenz (Widerspruchsfreiheit) der Daten. Um dieses Ziel zu erreichen bedarf es einer genauen Planung vom konzeptionellen Datenbankdesign (Informationsgewinnung, semantisches ER-Modell, logisches, normalisiertes Tabellenmodell) bis hin zum physischen Datenbankdesign mit Hilfe eines Datenbankmanagementsystems. In Gefahr ist die Konsistenz immer dann, wenn durch Benutzer oder das DBMS die Daten manipuliert, also verändert werden.[1]

Um diese Gefahr möglichst gering zu halten sind auf vier verschiedenen Ebenen sogenannte Integritätsregeln festzulegen:

- Bereichsintegrität: Attributwerte sind nur gültig, wenn sie einem bestimmten Wertebereich entsprechen. (z.B.: In einem Datumsfeld dürfen nur gültige Daten stehen – wird sichergestellt durch einen passenden Datentyp je Attribut)
- Entitätsintegrität: Jeder Datensatz ist eindeutig definiert. (wird sichergestellt durch ein Primärschlüsselattribut)
- Referentielle Integrität: Beziehungen zwischen Tabellen müssen synchronisiert bleiben.
- Benutzerdefinierte Integrität: Sonstige vom Benutzer festgelegte Regeln (z.B.: Datum darf nicht vor 01.01.2000 liegen)

5.4.1 Referentielle Integrität

Die referentielle Integrität (auch **Beziehungsintegrität**) besagt, dass Attributwerte eines Fremdschlüssels auch als Attributwert des Primärschlüssels vorhanden sein müssen damit die Konstistenz zwischen Tabellen, die über Primär-Fremdschlüsselbeziehungen verbunden sind, erhalten bleibt.
Ist dieser Regelsatz aktiviert hat dies folgende Auswirkungen:

1. Datensätze der referenzierenden Tabelle (auch Kind-Tabelle ≙ n-Seite der Beziehung) können nur dann gespeichert werden, wenn der Attributwert des Fremdschlüssels einmalig in der referenzierten Tabelle (auch Eltern-Tabelle ≙ 1-Seite der Beziehung) als Primärschlüsselwert vorhanden ist. Im Falle, dass ein referenzierter Wert nicht vorhanden ist, kann der Datensatz nicht gespeichert werden.
   Zum Beispiel kann ein Datensatz für einen neuen Auszubildendenden nur dann eingefügt werden, wenn die Klassennummer des Auszubildendenden in der Tabelle Klasse auch tatsächlich existiert. Andernfalls kommt es zu einer Fehlermeldung. Analoges gilt für Änderungen des Fremdschlüsselwertes bestehender Auszubildender!
2. Die Änderung des Wertes eines Primärschlüssels in der referenzierten Tabelle (1-Seite) ist standardmäßig nicht möglich, wenn dieser als Wert in einem Datensatz der referenzierenden Tabelle (n-Seite) existiert!
   Zum Beispiel kann die Klassennummer k_id mit dem Primärschlüsselwert '10' nicht geändert werden, wenn es einen Auszubildendenden gibt, der dieser Klasse über den Fremdschlüsselwert ‚10' zugeordnet ist. DBMS stellen allerdings Erweiterungen zur Verfügung, die dieses Standardverhalten ändern können:

Erweiterungen sind:

- Änderungsweitergabe
  Wenn der Wert des Primärschlüssels (k_id in der Tabelle Klasse) geändert wird, ändert sich automatisch der Wert des Fremdschlüssels (k_id in der Klasse Auszubildender).

[1] http://www.datenbanken-verstehen.de/datenbank-grundlagen/dbms/datenbankkonsistenz/ abgerufen am 12.05.18 -19:00 Uhr

- Löschweitergabe
  Wird ein Datensatz der referenzierten Tabelle (1-Seite) gelöscht, werden automatisch alle Datensätze mit entsprechendem Wert im Fremdschlüselattribut gelöscht. Zum Beispiel werden nach dem Löschen der Klasse '10' der Tabelle Klasse alle Auszubildendenden die der Klasse '10' zugeordnet sind automatisch gelöscht – zugegebenermaßen ein wenig sinnvolles Beispiel😉 .

Diese Funktionen können je nach DBMS optional gesetzt werden. Die referentielle Integrität ist dafür die Grundlage.

Aufgabe 7:

- Lassen Sie sich zunächst den aktuellen Inhalt der Tabellen Mitarbeiter und Abteilung anzeigen. Den entsprechenden Befehl finden Sie im ersten Beispiel des folgenden Abschnitts 6.1
- Überprüfen Sie, ob für die beiden Tabellen referentielle Integrität bereits aktiviert ist, in dem Sie versuchen, den Wert von IDAbteilung (Primärschlüssel) von Einkauf auf '99' ändern!
- Hat geklappt, oder? Das heißt die Regel für referentielle Integrität ist noch nicht aktiviert! Machen Sie die gerade vorgenommene Änderung rückgängig, in dem Sie den Wert wieder auf '1' zurücksetzen
- Aktivieren Sie die referentielle Integrität nun durch folgenden Befehl:

```
alter table Mitarbeiter #n-Seite der Beziehung!
add foreign key (AbteilungID)
references Abteilung(IDAbteilung);
```

  Erklärung zum Befehl:
  Der Fremschlüssel 'AbteilungID' der Tabelle Mitarbeiter (n-Seite) wird verknüpft (referenziert) mit dem Primärschlüssel IDAbteilung der Tabelle Abteilung (1-Seite)[2]
- Versuchen Sie, die Abteilung Einkauf aus der Tabelle Abteilung zu löschen (DELETE)!
  Notieren Sie die englische Fehlermeldung bis zum Wort 'fails':

[2] Interessierte können z.B. unter http://www.mysqltutorial.org/mysql-foreign-key/ genaueres nachlesen!

6 DQL – Data Query Language

MySQL als Abfragesprache

Die Data Query Language oder kurz DQL wird für die Abfrage von relationalen Datenbanken eingesetzt. Für die Abfragen verwendet man die SELECT-Anweisung.

6.1 Auswählen - SELECT-Anweisung

Mit der SELECT-Anweisung werden Datenwerte aus Tabellen zusammengestellt und ausgelesen. Das Ergebnis ist eine Tabelle. Mit dem SELECT-Statement können einfache (eine Tabelle) und komplexe Abfragen (zwei und mehr Tabellen) formuliert werden. Eine gültige Abfrage muss mindestens die Auswahl (SELECT) und die Angabe einer Tabelle (FROM) beinhalten.

Spalten auswählen - Projektion

Das Symbol '*' gibt alle Spalten (columns) aller verwendeten Tabellen aus. Will man nur bestimmte Spalten sehen, so gibt man ihre Namen durch Kommata getrennt in der gewünschten Reihenfolge an. Soll eine Spalte mit einer anderen Überschrift als dem Spaltennamen versehen werden, so hängt man die gewünschte Überschrift mit AS an den eigentlichen Spaltennamen an. Die neue Überschrift nennt man „Alias".

Syntax:

```
SELECT <Spalten>
FROM <Tabellenname>;
```

Beispiele:

```
SELECT *
FROM Auszubildender;
```

```
SELECT a_name, a_vorname, a_gebdat
FROM Auszubildender;
```

```
SELECT [DISTINCT] a_name AS "Auszubildender"
FROM Auszubildender;
```

Hinweis: Durch den optionalen Befehl DISTINCT werden Werte, die mehrfach auftreten, nur einmal angezeigt. Bei Verwendung von DISTINCT im obigen Beispiel würde der Name „Müller" also nur einmal angezeigt werden, obwohl mehrere Auszubildende mit dem Namen Müller existieren könnten.

Aufgabe 8:

- Lassen Sie sich zuerst alle Datensätze der Tabelle Abteilung anzeigen!
- Lassen Sie sich nur die nur die Abteilungsbezeichnungen anzeigen!

6.2 Zeilen auswählen – Selektion mit der WHERE-Klausel

Der optionale WHERE-Teil (Selektion) in einem SELECT-Statement gibt Bedingungen an, mit denen die Menge der angezeigten Datensätze weiter eingeschränkt werden kann. Auf Basis der Projektion werden nun die gewünschten Zeilen (rows) ausgewählt.

Syntax:

```
SELECT <Spalten>
FROM <Tabellenname>
[WHERE <Bedingungen>];
```

Beispiele:

```
SELECT *
FROM Auszubildender
WHERE a_id=2;
```

```
SELECT a_name, a_vorname, a_gebdat
FROM Auszubildender
WHERE a_gebdat > '1980-01-01';
```

6.3 Vergleichsoperatoren

=	gleich
<>	ungleich
>	größer als
<	kleiner als
>=	größer gleich
<=	kleiner gleich

Diese Bedingungen können durch AND, OR, NOT und Klammern zu komplexen Ausdrücken kombiniert werden:

Beispiele:

```
SELECT a_name, a_vorname
FROM Auszubildender
WHERE a_id=4 AND a_gehalt > 500.00;
```

```
SELECT a_name, a_vorname
FROM Auszubildender
WHERE a_geschlecht='m'
AND
 (a_ort='Attenkirchen' OR
  a_ort='Zolling' OR
  a_ort = 'Garmisch');
```

Aufgabe 9:

- Lassen Sie alle Produkte inkl. Preis des FoodShops anzeigen, die mindestens 5,- Euro kosten.
- Lassen Sie alle Kunden mit Sitz in Mainz anzeigen.

6.4 Spezielle Vergleichsoperatoren – LIKE, BETWEEN, IN

LIKE	Suche nach einem Muster mit Verwendung von Platzhaltern: '_' steht für genau ein beliebiges Zeichen, '%' steht für eine beliebige Anzahl von beliebigen Zeichen
BETWEEN … AND …	Wertebereich mit definierter Unter- und Obergrenze
IN	Liste von Werten durch Kommata getrennt

Hinweis: Die angegebenen Grenzwerte sind im Fall von BETWEEN ... AND … in der Auswahl miteingeschlossen!

Beispiele:

```
SELECT a_name, a_vorname
FROM Auszubildender
WHERE a_name LIKE 'H%';
```

```
SELECT a_name, a_vorname
FROM Auszubildender
WHERE a_name LIKE 'Mk_er'
AND a_ort IN ('Attenkirchen','Zolling','Garmisch');
```

```
SELECT a_name, a_vorname, a_gebdat
FROM Auszubildender
WHERE a_gebdat BETWEEN '1975-01-01' AND '1975-12-31';
```

6.5 Sortierung mit ORDER BY

Mit der ORDER BY-Klausel kann die Ergebnistabelle nach ein oder mehreren Kriterien sortiert werden. Durch Angabe von ASC wird aufsteigend, bei DESC absteigend sortiert. Fehlt die Angabe, wird immer aufsteigend sortiert.

Syntax:

```
SELECT <Spalten>
FROM <Tabellenname>
[WHERE <Bedingungen>]
[ORDER BY <Spalte> ASC | DESC];
```

Beispiele:

```
SELECT a_name, a_gehalt
FROM Auszubildender
WHERE a_gehalt > 250.00
ORDER BY a_gehalt DESC;
```

Aufgabe 10:

- Lassen Sie alle Produkte inkl. Preis des FoodShops anzeigen, die zwischen 5,- und 7,- Euro kosten. Die Ergebnistabelle soll das günstigste Produkt zuerst anzeigen.
- Sortieren Sie die Mitarbeiter nach der Höhe ihres Gehaltes (höchstes Gehalt zuerst).

6.6 Verdichtung - GROUP BY-Klausel:

Mit der GROUP BY-Klausel werden bestimmte Sätze zu einem Datensatz in der Ergebnistabelle zusammengefasst. Alle Datensätze, die die gleichen Werte, in den hinter GROUP BY angegebenen Spalten besitzen, werden zu einem Datensatz zusammengefasst.

Syntax:

```
SELECT <Spalten>
FROM <Tabellenname>
[WHERE <Bedingungen>]
[GROUP BY <Gruppierungsspalten>]
[ORDER BY <Spalte> ASC | DESC];
```

Beispiele:

```
SELECT a_vorname AS Vorname
FROM Auszubildender
GROUP BY a_vorname;
```

Dieses Statement gibt jeden Vornamen in der Tabelle ein einziges Mal aus. Wir haben allerdings für diese Auflistung schon eine andere Möglichkeit kennen gelernt, nämlich über das Schlüsselwort DISTINCT:

```
SELECT DISTINCT a_vorname AS Vorname
FROM Auszubildender;
```

6.7 Aggregationsfunktionen – COUNT, SUM, AVG, MAX, MIN

Im Zusammenhang mit Gruppierung spielen Aggregationsfunktion eine Rolle. Sie werden auf allen Datensätzen, die zu einer Gruppierung gehören, ausgeführt.

Mögliche Aggregatsfunktionen:

COUNT	Anzahl von Werten
SUM	Summe von Werten
AVG	arithmetischer Mittelwert von Werten(SUM / COUNT)
MAX	größter Wert
MIN	kleinster Wert

Beispiele:

```
SELECT count(a_id) AS 'Anzahl der Auszubildender'
FROM Auszubildender
GROUP BY k_id;
```

```
SELECT k_id, max(a_gehalt) AS 'Höchstes Gehalt je Klasse'
FROM Auszubildender
GROUP BY k_id;
```

```
SELECT SUM(a_gehalt) AS 'Gehaltssumme aller Auszubildender'
FROM Auszubildender;
```

```
SELECT SUM(a_gehalt) AS 'Gehaltssumme pro Klasse'
FROM Auszubildender
GROUP BY k_id;
```

6.8 HAVING-Klausel

Durch die WHERE-Klausel können nur Bedingungen zu Werten formuliert werden, die bereits in den Datensätzen vorhanden sind. Mit der HAVING-Klausel können allerdings auch Bedingungen mit Werten formuliert werden, die erst durch eine Aggregatfunktion berechnet werden.

Syntax:

```
SELECT <Spalten>
FROM <Tabellenname>
[WHERE <Bedingungen1>]
[GROUP BY <Gruppierungsspalten>]
[HAVING <Bedingungen2>];
```

Beispiele:

```
SELECT k_id AS KlassenID, COUNT(a_id) AS 'Anzahl Auszubildender'
FROM Auszubildender
GROUP BY k_id
HAVING COUNT(a_id) > 10;
```

Es wird zu allen Klassen, die mehr als 10 Auszubildendende haben, die Anzahl der Auszubildendenden angegeben.

Aufgabe 11: (HAVING-Klausel):

- Ermitteln Sie die Tage, an denen mehr als 5 Bestellungen eingegangen sind!

6.9 UNION – Vereinigung

Durch die UNION-Klausel können Datensätze aus zwei Tabellen zusammengeführt werden. Dies ist möglich mit Tabellen gleicher als auch unterschiedlicher Datenstruktur!
Tabellen mit gleicher Datenstruktur:

Syntax:

```
SELECT <Auswahl> FROM <Tabellenname1>
UNION [ALL]
SELECT <Auswahl> FROM <Tabellenname2>;
```

Wählen wir zunächst die Option UNION ALL:
Die Anweisung UNION ALL vereinigt die Ergebnismengen zweier Abfragen, wobei alle Werte, also auch mehrfach vorkommende Datensätze erhalten bleiben.
Beispiel:

```
SELECT * FROM Auszubildender
UNION ALL
SELECT * FROM Auszubildender_A;
```

Tabellen mit ungleicher Datenstruktur:
Vesucht man eine Abfrage mit UNION-Klausel über zwei Tabellen mit unterschiedlicher Datenstruktur (z.B. unterschiedliche Zahl von Attributen) auszuführen kommt es zu einer Fehlermeldung: `"The used SELECT-Statements have a different number of columns"`.
In diesem Fall hat man die Möglichkeit durch Auswahl einer gleichen Anzahl von Attributen die Abfrage auszuführen!

Beispiel:

```
SELECT a_name, a_vorname, a_gehalt FROM Auszubildender
UNION ALL
SELECT a_name, a_vorname, a_gehalt FROM mitarbeiter_A;
```

Aufgabe 12:

- Erstellen Sie eine neue Tabelle "produkt_neu":

```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| pnr   | char(3)     | NO   | PRI |         |       |
| bez   | varchar(30) | NO   |     |         |       |
| p_grp | varchar(30) | NO   |     |         |       |
| preis | decimal(9,2)| NO   |     |         |       |
+-------+-------------+------+-----+---------+-------+
```

- Fügen Sie die folgenden beiden Datensätze in die Tabelle produkt_neu ein:
  `100,'Tofu natur', 'Health food',3.99`
  `101,'Kichererbsen', 'Health food',1.99`
- Geben Sie alle Produkte der beiden Tabellen produkt und produkt_neu aus!

6.9.1 Die UNION-Klausel im Unterschied zur UNION ALL-Klausel

Nehmen wir die Option UNION allein, so werden in der Ergebnistabelle nur unterschiedliche Werte ausgewählt.

Beispiel:

```
SELECT a_gebdat FROM Auszubildender
UNION
SELECT aa_gebdat FROM Auszubildender_A;
```

Ergebnistabelle mit UNION ALL
a_gebdat
1975-12-17
1974-02-20
1984-02-22
1984-04-02
1950-01-12
1956-10-13
1953-03-31
1974-02-20

Ergebnistabelle mit UNION
a_gebdat
1975-12-17
1974-02-20
1984-02-22
1984-04-02
1950-01-12
1956-10-13
1953-03-31

6.10 JOIN – Verbindung

Abfragen über zwei oder mehr Tabellen. Dazu sehen wir uns die folgenden zwei Tabellen an:

Auszubildender					
a_id	a_name	a_vorname	a_gebdat	a_gehalt	k_id
1	Schmitt	Johann	1975-12-17	560.00	4
2	Albert	Max	1974-02-20	420.00	1
3	Wolf	Rita	1984-02-22	250.00	1
4	Jonas	Sofie	1984-04-02	595.00	2

projekt		
p_id	p_bez	p_leiter
1	InfoPlus	3
2	Office 2010	1
3	Custom 2.0	1
4	System Neu	4

Die folgenden zwei Aufgaben beziehen sich auf die beiden obigen Tabellen Auszubildender und projekt! Beantworten Sie die Aufgaben schriftlich in Ihrem Skript!

Aufgabe 13:

- Notieren Sie in für jedes Projekt den Namen und Vornamen des Projektleiters!

Projektname P-Leiter_Name P-Leiter_Vorname

Projekt 1: .....

Projekt 2: .....

Projekt 3: .....

Projekt 4: .....

6.11 CROSS JOIN - Kartesisches Produkt

Das kartesische Produkt ist ein Join, der nicht beschränkt ist. Alle Datensätze einer Tabelle werden mit allen Datensätzen der anderen Tabelle(n) kombiniert.

Beispiel:

```
SELECT * FROM Auszubildender
CROSS JOIN projekt;
```

Beispielhafte Ergebnistabelle – stimmt mit den Daten unserer aktuellen DB Betrieb nicht ganz überein!								
a_id	a_name	a_vorname	a_gebdat	a_gehalt	k_id	p_id	p_bez	p_leiter
1	Schmitt	Johann	1975-12-17	560.00	4	1	InfoPlus	3
1	Schmitt	Johann	1975-12-17	560.00	4	2	Office 2010	1
1	Schmitt	Johann	1975-12-17	560.00	4	3	Custom 2.0	1
1	Schmitt	Johann	1975-12-17	560.00	4	4	System Neu	4
2	Albert	Max	1974-02-20	420.00	1	1	InfoPlus	3
2	Albert	Max	1974-02-20	420.00	1	2	Office 2010	1
2	Albert	Max	1974-02-20	420.00	1	3	Custom 2.0	1
2	Albert	Max	1974-02-20	420.00	1	4	System Neu	4
3	Wolf	Rita	1984-02-22	250.00	3	1	InfoPlus	3
3	Wolf	Rita	1984-02-22	250.00	3	2	Office 2010	1
3	Wolf	Rita	1984-02-22	250.00	3	3	Custom 2.0	1
3	Wolf	Rita	1984-02-22	250.00	3	4	System Neu	4
4	Jonas	Sofie	1984-04-02	595.00	2	1	InfoPlus	3
4	Jonas	Sofie	1984-04-02	595.00	2	2	Office 2010	1
4	Jonas	Sofie	1984-04-02	595.00	2	3	Custom 2.0	1
4	Jonas	Sofie	1984-04-02	595.00	2	4	System Neu	4

Diese Ergebnistabelle macht natürlich wenig Sinn. Daher werden Einschränkungen vorgenommen. Es wird untersucht, ob Attribute mit identischen Werten vorkommen. Dies ist dann der Ausgangspunkt für das folgende Konzepte des JOIN.

Aufgabe 14:

- Markieren Sie diejenigen Datensätze der obigen Ergebnistabelle, welche richtige Zuordnungen von Projektleitern zu Projekten anzeigen!

- Welche Besonderheit fällt Ihnen bei den markierten Datensätzen auf, die Ihnen geholfen hat, die vier richtigen Datensätze zu markieren?

.....

.....

6.12 INNER JOIN (EQUI-JOIN)

Tabellen werden in einem JOIN über die Schlüsselattribute (Primärschlüssel zu Fremdschlüssel) verknüpft, die in beiden Tabellen vorkommen. Auf diese Art kann man in einer SQL-Abfrage zwei oder mehr Tabellen miteinander verknüpfen. Dieser JOIN heißt auch EQUI-JOIN, weil er auf der Gleichheit von Werten zweier Spalten (Primärschlüsselwert und Fremdschlüsselwert) basiert.

Im folgendenden Beispiel wird nun gesucht, welcher Auszubildende ein Projekt leitet.
Dazu benötigt man Informationen aus den beiden Tabellen `Auszubildender` und `Schulprojekt`. Die beiden Tabellen müssen über `Primärschlüsselwert (a_id) = Fremdschlüsselwert (p_leiter)` verknüpft werden.

```
SELECT *                          #Alternative a) INNER JOIN
FROM Auszubildender
INNER JOIN schulprojekt
ON a_id=p_leiter;
```

```
SELECT *                          #Alternative b) NATURAL JOIN
FROM Auszubildender, schulprojekt
WHERE a_id=p_leiter;
```

Als Ergebnis resultiert folgende Tabelle:

Ergebnistabelle								
a_id	a_name	a_vorname	a_gebdat	a_gehalt	k_id	p_id	p_bez	p_leiter
1	Schmitt	Johann	1975-12-17	560.00	4	2	Office 2010	1
1	Schmitt	Johann	1975-12-17	560.00	4	3	Custom 2.0	1
3	Wolf	Rita	1984-02-22	250.00	3	1	InfoPlus	3
4	Jonas	Sofie	1984-04-02	595.00	2	4	System Neu	4

Auch bei Abragen über mehere Tabellen kann die Auswahl der angezeigten Tabellen eingeschränkt werden:

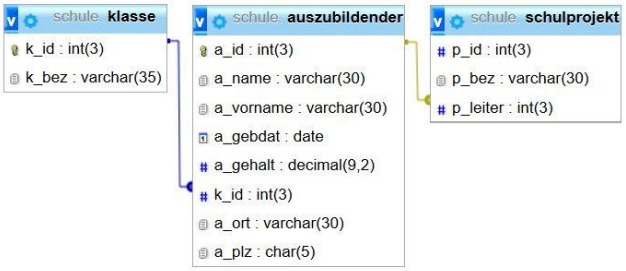
Beispiel:

```
SELECT a.a_id, a.a_name, p.p_bez
FROM Auszubildender a
INNER JOIN schulprojekt p
ON a.a_id=p.p_leiter;
```

Ergebnistabelle		
a_id	a_name	p_bez
1	Schmitt	Office 2010
1	Schmitt	Custom 2.0
3	Wolf	InfoPlus
4	Jonas	System Neu

Im nächsten Beispiel wird nun die Klassenbezeichnung derjenigen Auszubildenden gesucht, die ein Projekt leiten!

Für diese Abfragen benötigt man Informationen aus drei! Tabellen, welche miteinander verknüpft werden müssen.



Alternative a – INNER JOIN

```
SELECT k.k_bez, a.a_name, p.p_bez
FROM Klasse k
INNER JOIN Auszubildender a
ON k.k_id = a.k_id
INNER JOIN schulprojekt p
ON a.a_id=p.p_leiter;
```

Alternative b – Natural Join:

```
SELECT k.k_bez, a.a_name, p.p_bez
FROM Klasse k, Auszubildender a, schulprojekt p
WHERER k.k_id = a.k_id
AND a.a_id=p.p_leiter;
```

Ergebnistabelle		
k_bez	a_name	p_bez
2FS133	Schmitt	Office 2010
2FS133	Schmitt	Custom 2.0
3FA073	Wolf	InfoPlus
4KA331	Jonas	System Neu

6.13 Aliase von Tabellennamen

In obigem Beispiel wurden für die beiden Tabellen Auszubildender und projekt die Aliasnamen (Abkürzungen) 'm' und 'p' definiert. Aliasnamen sind Namen, die man hinter den eigentlichen Tabellennamen schreibt. Man kann sie dann in der ganzen Abfrage gleichwertig dem Tabellennamen einsetzen.
Kommen Spalten mit demselben Namen in mehreren Tabellen vor, so muss zur eindeutigen Identifizierung der Tabellenname oder eben der Aliasname vor den Spaltennamen gestellt werden:
<tabellenname>.<spaltenname> oder <aliasname>.<spaltenname>

Aufgabe 15: (INNER JOIN):

- In welcher Abteilung arbeitet Heinz-Heribert Koch?
- Welche Mitarbeiter arbeiten in der Abteilung Produktion?
- Geben Sie aus, wie viele Bestellungen jeder einzelne Kunde aufgegeben hat!
- Wie oft hat die "Tam-Tam GmbH" bestellt?
- Welche Produkte hat die "Tam-Tam GmbH" bestellt?
- Geben Sie die Kunden und die Anzahl bisher getätigter Bestellungen aus! Der Kunde mit den meisten Bestellungen soll zuerst ausgegeben werden!
- Welcher Mitarbeiter (Name, Vorname) hat die Bestellung der "Tam-Tam GmbH" vom 3. Oktober 2001 bearbeitet?
- Geben Sie alle Gruppenleiter des Foodshops mit Namen, Funktionsnummer und Funktionsbezeichnung aus!
- Geben Sie zusätzlich aus, in welcher Abteilung die Gruppenleiter arbeiten! (Abfrage über drei! Tabellen)

6.14 LEFT OUTER JOIN

Wiederum wird die Verbindung von zwei Tabellen über den Primärschlüssel der einen Tabelle, der zum Fremdschlüssel in der anderen Tabelle wird, hergestellt.
Im Beispiel werden nun alle Auszubildendenden gesucht, die ein Projekt leiten. Es sollen aber auch diejenigen Auszubildendenden aufgelistet werden, die kein Projekt leiten. Somit sollen alle Datensätze aus der ("linken") Tabelle Auszubildender herausgesucht werden und nur die Datensätze aus der "rechten" Tabelle projekt, die die angegebene Bedingung erfüllen. Hat ein Auszubildender kein Projekt, steht dann an entsprechender Stelle NULL.
Beispiel:

```
SELECT a.a_id, a.a_name, p.p_bez
FROM Auszubildender a
LEFT [OUTER] JOIN project p
ON a.a_id=p.p_leiter;
```

Ergebnistabelle		
a_id	a_name	p_bez
1	Schmitt	Custom 2.0
2	Albert	NULL
3	Hof	InfoPlus
4	Jonas	System Neu

Der RIGHT OUTER JOIN funktioniert entsprechend.

Aufgabe 16:

- Welche Mitarbeiter haben Bestellungen bearbeitet? Geben Sie aus: Mitarbeiternummer, Mitarbeitername (INNER JOIN)
- Ändern Sie die Anweisung, damit auch alle Mitarbeiter angezeigt werden, denen keine Bestellung zugeordnet werden kann! (LEFT JOIN)
- Ändern Sie die Anweisung, damit nur Mitarbeiter angezeigt werden, die keine Bestellungen bearbeitet haben! (LEFT JOIN)

7 DCL – Data Control Language

Vorab: Dieser Abschnitt ist nicht prüfungsrelevant und wird hier lediglich der Vollständigkeit halber erwähnt!

SQL als Kontrollsprache
Die Data Control Language (DCL) enthält die Befehle zur Sicherheit einer Datenbank.
COMMIT: führt die Änderungen an einer Datenbank endgültig durch
ROLLBACK: solange eine Änderung noch nicht mit COMMIT abgeschlossen ist, kann der ursprüngliche Zustand der Datenbank mit ROLLBACK wieder hergestellt werden
GRANT: erlaubt dem Nutzer bestimmte Aktionen an der Datenbank
REVOKE: entzieht mit GRANT verliehene Zugriffsrechte

7.1 Transaktionen

Die Elemente der DCL werden als Befehle der Transaktionsumgebung bezeichnet. In einer Transaktionsumgebung wird für jede Änderungen an den Daten eine Transaktion eröffnet, nach Abschluss der Änderung quittiert das DBMS den Erhalt der Daten, mittels COMMIT wird die Transaktion bzw. Änderung abgeschlossen. Kommt es während der Bearbeitung zu einem System- oder Netzwerkausfall oder werden die Daten unvollständig übertragen, wird das DBMS angewiesen einen ROLLBACK durchzuführen.
Eine Transaktion führt eine Datenbank von einem konsistenten Zustand immer in einen konsistenten Zustand. Das bedeutet, sie wird entweder vollständig oder gar nicht ausgeführt. Wird im Zuge der Transaktion festgestellt, dass sie teilweise nicht ausgeführt werden kann, wird der Rollback eingeleitet. Dieser sorgt dafür, dass die Datenbank in dem Zustand kommt, in dem sie von der Transaktion vorgefunden wurde. Ist jeder Teilaspekt der Transaktion einwandfrei durchlaufen, wird per Commit die Änderung endgültig gemacht. Die rollback-Anweisung macht alle Änderungen an der Datenbank rückgängig, die der Benutzer seit dem letzten Commit oder seit dem Beginn der Sitzung vorgenommen hat.
Unter einer Transaktion versteht man die Zusammenfassung einer Menge von SQL-Anweisungen zu einer einzigen logischen Anweisung. Man kann sich dies so ähnlich vorstellen wie einen BEGIN-END-Block in einem Programm, dem eine IF-Anweisung vorausgeht. Transaktionen haben bestimmte Regeln zu erfüllen; man spricht auch vom ACID-Prinzip. Zur Sicherstellung dieser Regeln arbeiten Datenbanksysteme mit Sperrmechanismen.

Atomicity	Eine Transaktion ist atomar, sie wird wie eine einzige Anweisung behandelt.
Consistency	Die Datenbank ist vor oder nach der Ausführung einer Transaktion in einem konsistenten Zustand.
Isolation	Eine Menge zeitlich verzahnter Transaktionen wird so ausgeführt, als ob jede Transaktion als einzige - isoliert - ausgeführt wird (Serialisierbarkeit).
Durability	Die von einer erfolgreichen Transaktion durchgeführten Änderungen verbleiben dauerhaft in der Datenbank und dürfen auch durch einen Systemabsturz nicht verloren gehen.

Die in der Transaktion gemachten Änderungen können also nur komplett festgeschrieben oder komplett zurückgerollt werden.

COMMIT
macht die Änderungen, die seit Beginn der Transaktion durchgeführt wurden, permanent
ROLLBACK
führt ein undo seit Beginn einer Transaktion oder seit einem Savepoint durch
SAVEPOINT
Setzen eines Punktes, bis zu dem ein Rollback durchgeführt werden kann. Bei langlaufenden Transaktionen muss im Fehlerfall nicht bis zum Anfang der Transaktion zurückgerollt werden.
SET SAVEPOINT
Setzen von Eigenschaften für die aktuelle Transaktion

7.2 Zugriffsrechte

Jede Datenbank hat eine eigene Benutzer- und Rechteverwaltung. Benutzer sind grob eingeteilt in:
System-Administrator DBA (meist einer)
Datenbankbesitzer DBO (meist mehrere)
gewöhnliche Benutzer (meist viele).
Grundsätzlich gilt für alle Benutzer (außer DBA), dass sie nur die Operationen ausführen dürfen, für die sie in einer Systemtabelle eingetragen sind. Alle Rechte innerhalb einer Datenbank werden über die GRANT Anweisung vergeben und über die REVOKE Anweisung entzogen.
Folgende Rechte können vergeben werden:

SELECT - Erteilt das Recht, Sätze aus einer Tabelle zu selektieren. Hierbei kann optional eine Liste von Spalten angegeben werden, um das SELECT Recht auf diese Spalte zu begrenzen.

UPDATE - Erteilt das Recht, Sätze aus einer Tabelle zu ändern. Wie beim SELECT Recht kann optional eine Liste von Spalten angegeben werden, für die das UPDATE Recht gelten soll.

INSERT - Erteilt das Recht, Sätze in eine Tabelle einzufügen.

DELETE - Erteilt das Recht, Sätze aus einer Tabelle zu löschen.

ALTER - Erteilt das Recht, eine Tabelle durch einen ALTER TABLE zu ändern.

INDEX - Erteilt das Recht, Indizes auf eine Tabelle anzulegen.

REFERENCES - Erteilt das Recht, Spalten einer Tabelle in einem FOREIGN KEY zu referenzieren. Wie beim SELECT kann optional eine Liste von Spalten angegeben werden, die referenziert werden dürfen.

USAGE - Erteilt das Recht, eine Domain zu verwenden.

TRIGGER - Erteilt das Recht, einen Trigger für die Tabelle zu definieren.

EXECUTE - Erteilt das Recht, eine Routine auszuführen.

Syntax:
GRANT privileges ON object-name TO grantee [WITH GRANT OPTION]

privileges ::= { ALL [PRIVILEGES] | action [, ...] }

action ::=
DELETE
| ALTER
| INSERT
| UPDATE [(column-name-list)]
| SELECT [(column-name-list)]
| INDEX
| REFERENCES [(column-name-list)]
| TRIGGER
| USAGE
| EXECUTE

object-name ::=
[TABLE] table-name
| DOMAIN domain-name
| { PROCEDURE | FUNCTION } routine-name

GRANT RESOURCE TO grantee

GRANT DATABASE TO grantee

grantee ::= { user-authorization-identifier [, ...] | PUBLIC | ALL }

REVOKE
Durch die REVOKE Anweisung werden bestimmte Zugriffsrechte (object-privileges) auf eine Tabelle, View, Prozedur oder Datenbank einem oder mehreren Benutzern eines DBS entzogen. Die REVOKE Anweisung wird nur wirksam, wenn die angegebenen object-privileges vorher mit einer GRANT Anweisung vergeben wurden.
REVOKE privileges ON object-name FROM grantee

REVOKE RESOURCE FROM grantee

REVOKE DATABASE FROM grantee

DENY
Die DENY-Anweisung untersagt einem Benutzer oder einer Gruppe, etwas zu tun. Auch wenn keine expliziten Rechte da sind, sondern über Gruppenzugehörigkeiten Rechte ererbt werden, ist diese Tätigkeit untersagt.