Inhalt

Abbildungs- und Tabellenverzeichnis

1 Einführung in Datenbanksysteme

Was ist eine Datenbank?

Fast jeder kennt Excel und hat damit in seinem Leben schon einmal gearbeitet. In Excel gibt es Arbeitsblätter, die aus vielen Zellen bestehen, in die man verschiedene Werte (Zahlen, Text, Datümer, …) eintragen kann. Diese Zellen sind in Spalten und Zeilen organisiert.

Wenn man sich jetzt in Excel eine einfache Adressverwaltung aufbauen möchte, würde man vermutlich in verschiedene Spalten diverse Daten schreiben, z.B.:

- Vorname
- Nachname
- Straße
- PLZ
- Ort

In die Zeilen würden die Daten einer Adresse geschrieben werden. Das Ganze könnte in etwa folgendermaßen aussehen:

Vorname	Nachname	Straße	PLZ	Ort
Hugo	Schmidt	Sylter Weg 15	24145	Kiel
Bert	Meier	Schanzenstraße 1	20357	Hamburg
Egon	Müller	Exerzierplatz 3	24103	Kiel
Ivonne	Müller	Oldendorfer Weg 22	25524	Itzehoe

Damit haben wir eine Adresstabelle erstellt. Im Bereich der Datenbanken gibt es auch Tabellen, sog. ***Datenbanktabellen***. Jede Tabelle besteht aus mehreren ***Spalten***, die verschiedene Daten speichern. Die Zeilen heißen in Datenbankterminologie Datensätze. Ein ***Datensatz*** beinhaltet alle Spalten, die zusammen gehören. Im obigen Beispiel also alle Spalten zu einer Adresse. In Tabellen werden je nach Anwendungszweck unterschiedlichste Daten gespeichert.

In der Praxis benötigt man dann auch in der Regel mehr als eine Tabelle (in Excel ja auch), z.B. eine für Kunden, eine für verkaufte Produkte und eine für die Rechnungen. Alle drei Tabellen zusammen werden vielleicht benötigt, um die Auftragsabrechnung eines Pizzalieferdienstes durchzuführen, aber sie beinhalten unterschiedliche Daten. Wenn man mehrere Tabellen zusammenfasst, spricht man von einer ***Datenbank***. Diverse Datenbanken für unterschiedliche Anwendungsgebiete werden dann in einem ***Datenbankmanagementsystem (DBMS)*** verwaltet. Bekannte Vertreter sind hier z.B. Oracle, Microsoft SQL Server, IBM DB2.

Um die Daten in den Tabellen auslesen zu können, gibt es eine sogenannte Abfragesprache. Mit Hilfe diese Sprache kann man der Datenbank mitteilen, welche Datensätze man benötigt. Im Falle der Adresse möchte man vielleicht nur die Adresse von Hugo Schmidt wissen, um ihm eine Rechnung zu schreiben. Das kann man dann mit Hilfe der ***Structured Query Language (SQL)*** machen. Ein einfaches Beispiel sähe so aus:

```
SELECT *
FROM TBL_ADRESSEN
WHERE Nachname='Schmidt'
```

Dadurch würde man alle Spalten der Datensätze, bei denen der Nachname ‚Schmidt' ist zurück geliefert bekommen. Der Vorteil einer solchen Sprache ergibt sich vor allem bei Tabellen mit mehreren Millionen Datensätzen oder wenn man mehrere Tabellen zusammen abfragt (dazu in späteren Kapiteln mehr). Damit haben wir in diesem Kurs unser allererstes SQL verwendet. Viele weitere werden noch folgen. ;)

Neben SQL als Abfragesprache gibt es noch die ***Data Definition Language (DDL)*** und die ***Data Manipulation Language (DML)***. Mit Hilfe der DDL wird die Struktur der Datenbank erstellt, d.h. die Tabellen und ihre Strukturen. Darüber hinaus gibt es noch zahlreiche weitere Objekte in einer Datenbank (Views, Indizes, …), auf die in diesem Kurs aber nur am Rande eingegangen werden kann. Die DML dient dazu, die Tabellen mit Daten zu füllen bzw. diese zu ändern oder zu löschen.

Beziehungen zwischen Tabellen

In den meisten Datenbanken gibt es diverse Tabellen, um die Daten zu strukturieren und Redundanzen zu vermeiden (siehe auch Kapitel über Normalisierung).

Die verschiedenen Tabellen stehen dabei in ***Beziehung*** zueinander. Wenn man z.B. die Kunden in einer Tabelle hat und die Bestellungen in einer anderen, dann würde man in der Bestellungen-Tabelle nur noch die Kunden-Nr. ablegen und nicht die gesamten Informationen zu einem Kunden. Dadurch dass man diese Kunden-Nr. auch in der Kunden-Tabelle hat, kann man beide Tabellen in Beziehung setzen. Alle Daten zu einem Kunden (z.B. seine Adresse und Telefon-Nr.) werden nur in der Kunden-Tabelle gepflegt und nicht mehrfach in der Datenbank abgelegt. Mittels SQL kann man dann z.B. ermitteln, wie viele Bestellungen ich mit Kunden aus eine bestimmten Ort gemacht habe, einfach indem man die Kunden-Tabelle (mit dem Wohnort) und die Bestellungs-Tabelle miteinander verknüpft. Wenn sich jetzt die Adresse ändert, muss diese nur an einer einzigen Stelle angepasst werden.

In einem ***ER-Modell*** (ER = Entity Relationship) werden die Tabellen mit ihren Spalten und Beziehungen untereinander dargestellt. Ein solches Modell sieht z.B. wie folgende aus:

Abb.1: Einfaches ER – Modell

Die Beziehungen oder Verknüpfungen zwischen Tabellen erfolgen über sog. Schlüssel. Man unterscheidet dabei zwischen Primär- und Fremd-Schlüsseln. ***Primärschlüssel*** identifizieren einen Datensatz einer Tabelle eindeutig. Im Falle der Kunden-Tabelle wäre das z.B. die Kunden-Nr. Dort

sieht man in der Abbildung auch ein „P“, was den Primärschlüssel kennzeichnet. Ein ***Fremdschlüssel*** stellt einen Verweis auf den Primärschlüssel einer anderen Tabelle dar (gekennzeichnet durch „F“ in der obigen Abbildung), d.h. welcher Datensatz aus der anderen Tabelle mit dem der aktuellen verknüpft wird. Oder in unserem Beispiel: Welcher Kunden-Datensatz (Primär-Schlüssel) mit welcher Bestellung verknüpft (über den Fremd-Schlüssel) wird. Beziehungen werden also immer über die Kombination eines Fremd-Schlüssels mit einem Primär-Schlüssel definiert.

Je nachdem wie die Verhältnisse zwischen den Tabellen sind, unterscheidet man verschiedene Arten von Beziehungen:

1. 1:1 – Beziehung
2. 1:n – Beziehung
3. N:m – Beziehung
4. Rekursive Beziehungen

Bei der ***1:1 Beziehung*** existiert genau ein Datensatz in der Fremd-Tabelle zu jedem Datensatz in der Haupt-Tabelle und umgekehrt. Bei der ***1:n Beziehung*** existiert zu jedem Datensatz in der Fremd-Tabelle 1 bis n Datensätze in der Haupttabelle. Und zu jedem Datensatz in der Haupttabelle existiert genau 1 Datensatz in der Fremdtabelle. Diesen Fall trifft man in der Praxis am häufigsten an. In unserem Beispiel existiert eine 1:n Beziehung zwischen Kunden und Bestellungen. Jeder Kunde kann mehrfach in den Bestellungen auftauchen. Zu jeder Bestellung existiert genau 1 Kunde. ***N:m Beziehung***en bedeuten dann, dass zu jedem Datensatz in der einen Tabelle n Datensätze in der anderen Tabelle gehören. In relationalen Datenbanken kann dieser Umstand nur durch eine Zwischentabelle modelliert werden.

Man kann in den meisten DBMS einstellen, dass die ***referentielle Integrität*** vom System sichergestellt wird. Das bedeutet, dass zu einem Fremdschlüssel immer der entsprechende Datensatz in der referenzierten Tabelle existieren muss. Es wäre dann z.B. nicht möglich diesen Datensatz in der Fremd-Tabelle zu löschen, solange auf diesen in irgendeinem anderen Datensatz referenziert wird. Wenn man natürlich alle diese Datensätze zuerst löscht, dann kann auch der Satz in der Fremdtabelle gelöscht werden. Umgekehrt gilt das Gleiche. Man müsste erst diesen Satz in der Fremdtabelle erzeugen, damit man von anderen Tabellen auf diesen Datensatz referenzieren kann.

Um die referentielle Integrität vom DBMS ständig prüfen zu lassen, muss man sog. ***Foreign Key Constraints*** einrichten. Das ist nichts anderes als die Definition der Fremd-Schlüssel-Beziehung zwischen zwei Tabellen auf Datenbank Ebene. Es gibt noch sog. ***Check-Constraints***. Mit diesen kann sichergestellt werden, dass nur bestimmte Werte in bestimmte Spalten eingegeben werden können. Z.B. nur die Anreden ‚Herr‘ und ‚Frau‘ in die Spalte Anrede.

Normalformen

Damit Daten einfach und korrekt ausgewertet werden können, sollten diese immer folgende Eigenschaften haben:

- redundanzfrei
- eindeutig
- in sich konsistent

Um diese Eigenschaften der Daten dauerhaft sicherzustellen, gibt es bestimmte Regeln, die in Datenmodellen eingehalten werden sollen. Das sind die sog. ***Normalformen***. Es gibt 5

Normalformen, die jeweils bestimmte Redundanzen, die in den Daten durch das Datenmodell entstehen können, beseitigen. In der Praxis sind vor allem die ersten drei Normalformen relevant. Diese werden im Folgenden detailliert beschrieben.

Als Beispiel dient uns folgende Tabelle mit Kontakt-Informationen:

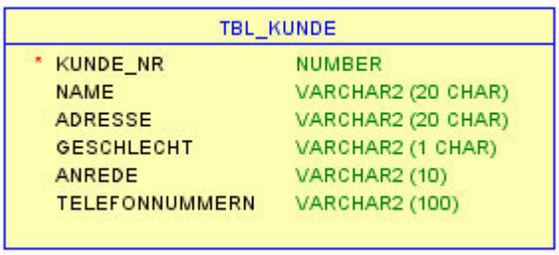

Abb.2: Bsp. Kundendaten - Ausgangslage

Eine Tabelle befindet sich in der ***1. Normalform***, wenn es keine Wiederholungsgruppen innerhalb der Tabelle gibt und jedes Attribut atomar (nicht in weitere Attribute aufteilbar) ist.

Auf unser Beispiel angewendet bedeutet das, dass es z.B. kein Attribut „Adresse“ geben kann, da dieses in Strasse, PLZ und Ort aufgeteilt werden kann (Atomarität). Wenn es dann noch mehrere Telefonnummern zu einem Kontakt geben kann, dann dürfen diese nicht in einem Feld Telefonnummer gespeichert werden (Wiederholungsgruppe) und auch nicht in Feldern Telefonnummer1 … TelefonnummerX (Wiederholungsgruppe), sondern es könnte ein Attribut TELEFONNUMMER ergänzt werden und dieses ist Teil des Schlüssels (Zusammengesetzt aus KONTAKT_NR und TELEFONNUMMER).

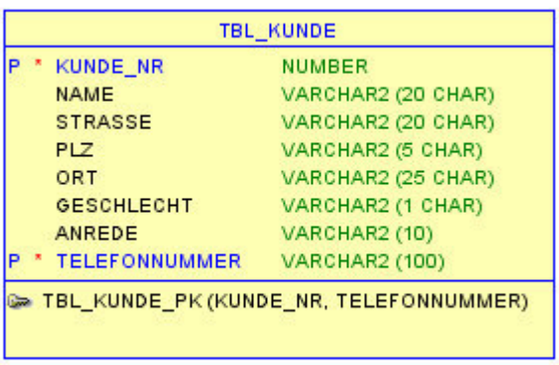

Abb.3: Bsp. Kundendaten – 1. Normalform

Durch Sicherstellung der 1. NF werden Daten überhaupt erst auswertbar. Wenn man z.B. die Adresse komplett in einem Attribut ablegt, wäre es schwierig bis unmöglich z.B. nach Ort zu sortieren und zu filtern.

Die ***2. Normalform*** ist gegeben, wenn die Tabelle in 1. NF vorliegt und jedes Nicht-Schlüsselfeld vom gesamten Schlüssel abhängt und nicht nur von Teilen des Schlüssels. Dadurch ist sichergestellt, dass

nur zusammengehörige Daten in einer Tabelle vorliegen. Außerdem werden Inkonsistenten vermieden, da Attribute nur einmalig vorkommen können.

Für unser Beispiel bedeutet das, dass NAME, STRASSE, ORT, GESCHLECHT und ANREDE nur von der KONTAKT_NR abhängen, jedoch nicht von der TELEFONNUMMER. Bei Anlage mehrere Telefonnummern zu einem Kontakt, würden die Kontakt-Informationen redundant in der Tabelle vorkommen, was 1. Speicherplatz verbraucht (heute nicht mehr das große Problem) und 2. Zu Inkonsistenzen führen kann. Man würde jetzt eine zusätzliche Tabelle mit den Telefonnummern erstellen.

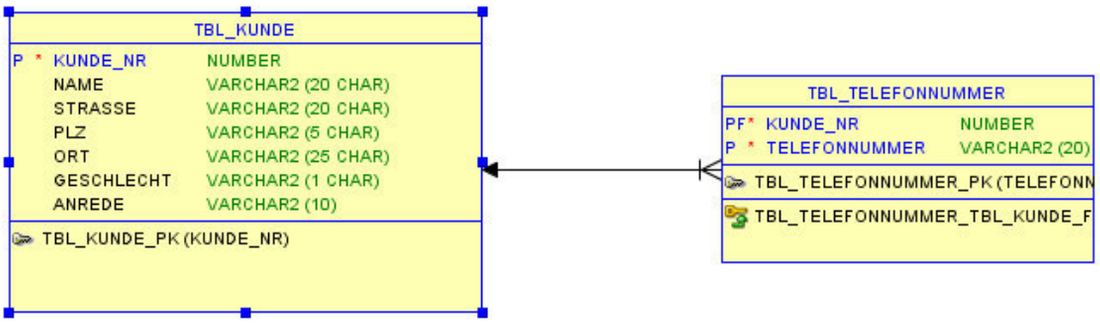

Abb.4: Bsp. Kundendaten – 2. Normalform

Zur Einhaltung der ***3. Normalform***, muss eine Tabelle in 2. NF vorliegen und Nicht-Schlüsselfelder dürfen nicht von anderen Nicht-Schlüsselfeldern abhängen. Durch die 3.NF werden weitere Redundanzen vermieden.

Das würde in unserem Beispiel bedeuten, dass man die Felder Ort und Anrede auslagern würde, weil der Ort von der PLZ abhängig ist und die Anrede vom Geschlecht. Auch dieses dient der Minimierung von Redundanzen, da diese zu Inkonsistenzen führen können

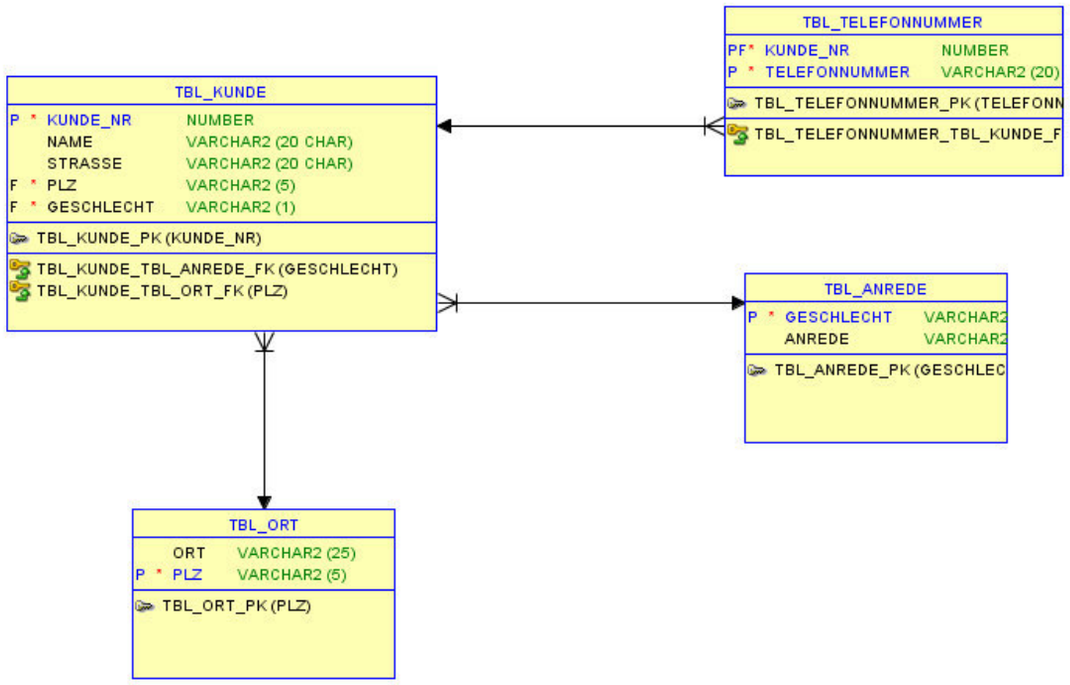

Abb.5: Bsp. Kundendaten – 3. Normalform

Durch die Normalisierung erhöht sich die Anzahl der Tabelle in einem Datenmodell. Das führt u.U. dazu, dass Abfragen länger laufen, da die Tabellen miteinander verknüpft werden müssen. Wenn die Antwortzeiten zu lang werden, kann man Datenmodelle auch wieder ***denormalisieren***. Dieses ist z.B. in Datawarehouse Umgebungen der Normalfall. Dort liegen die Tabellen im sog. Star-Schema vor.

Datenbanktransaktionen

Wenn Daten in Tabellen eingefügt oder verändert werden, muss sichergestellt werden, dass der Datenbestand hinterher auch noch in einer konsistenten Form vorliegt. Dazu existiert ein Paradigma im Datenbankbereich, welches ***ACID*** heißt:

- ***A***tomicity
- ***C***onsistancy
- ***I***solation
- ***D***urability

Dieses bedeutet, dass Datenänderungen so durchgeführt werden, dass sie ***atomar*** ausgeführt werden, d.h. dass ein Satz von Operationen ganz oder gar nicht ausgeführt wird. Weiterhin muss gewährleistet sein, dass nach der Ausführung, der Datenbestand weiterhin in sich ***konsistent*** ist (so lange er es vorher auch schon war). Die Änderungen sollen ***isoliert*** durchgeführt werden, so dass sich verschiedene Operationen nicht gegenseitig beeinfluss (z.B. gleichzeitiges löschen und lesen eines Datensatzes). Die geänderten Daten müssen ***dauerhaft*** gespeichert werden zum Abschluss.

Damit all dieses sichergestellt ist, gibt es sogenannte ***Transaktionen*** in modernen DBMS. Eine Transaktion stellt eine Klammer um eine oder mehrere DML (Insert, Update, Delete) – Anweisungen dar und wir als Block behandelt. Erst wenn alle Einzelanweisungen erfolgreich durchgeführt werden, wird das Ergebnis physisch festgeschrieben. Am Ende einer Transaktion kann diese entweder

festgeschrieben werden mittels ***COMMIT*** oder zurückabgewickelt werden mittels ***ROLLBACK***. Die Datenänderungen werden erst nach erfolgtem COMMIT sichtbar.

Aufgaben

1. Bringen Sie folgende Datenstruktur in die 3.NF. Es werden nun die Spalten aufgelistet:

Land	**ISO Code**	**Bundes-land**	**Stadt**	**Filiale**	**Mitarbeiter**	**Kontaktdaten**
Deutschland	DEU	Hamburg	Hamburg	Spitaler Straße	Hr. Schmidt Fr. Müller Hr. Meyer ….	Tel: 040-1234 Fax.: 040-1235 eMai: Hamburg@Firma.de
Dänemark	DEN	n/a	Kopenhagen	….	Fr. Sörensen ….	…
….						

2. Entwickeln Sie aus folgenden Anforderungen ein auswertbares Datenmodell in 3. NF
   - Es sollen CD-Titel gespeichert werden (Name)
   - Dazu sollen Interpreten und Alben gespeichert werden
   - Pro Album sollen die Titel ausgewertet werden können

4 Einfache Abfragen erstellen

Spalten selektieren (SELECT)

In diesem Kapitel werden wir die ersten einfachen Abfragen erstellen. Wir werden nur Daten aus einer einzelnen Tabelle abfragen und nach und nach die grundlegende Struktur eines SQL Statements kennen lernen. Das einfachste Beispiel sieht folgendermaßen aus:

```
SELECT *
FROM TBL_KUNDE
```

Ergebnis:

	KUNDE_NR	VORNAME	NACHNAME	STRASSE	PLZ	ORT	GESCHLECHT	GEB
1	1	Horst	Huber	.....	20357	Hamburg	M	.....
2	2	Erika	Schmidt	.....	22512	Hamburg	F	.....
3	3	Bert	Müller	.....	22123	Hamburg	M	.....
4	4	Hubertus	Meyer-Huber	.....	24106	Kiel	m	.....
5	5	Hanna	von Bergmann	.....	24145	Kiel	f	.....
6	6	Tobias	Maier	.....	26105	Flensburg	M	.....
7	7	Fabian	Lindemann	.....	23809	Lübeck	m	.....

Der Stern steht somit stellvertretend für alle Spalten der Tabelle. Mittels FROM <Tabellenname> wird die Tabelle definiert, aus der die Daten selektiert werden. Wenn wir nur bestimmte Spalten anzeigen wollen, können wir diese direkt mit Namen ansprechen und mehrere Spalten durch Kommata getrennt angeben:

```
SELECT KUNDE_NR,
       VORNAME,
       NACHNAME
FROM TBL_KUNDE
```

Das Ergebnis dazu besteht nur noch aus drei Spalten:

Ergebnis:

	KUNDE_NR	VORNAME	NACHNAME
1	1	Horst	Huber
2	2	Erika	Schmidt
3	3	Bert	Müller
4	4	Hubertus	Meyer-Huber
5	5	Hanna	von Bergmann
6	6	Tobias	Maier
7	7	Fabian	Lindemann

Wenn wir jetzt nur noch die Spalte ORT abfragen, würde das Ergebnis folgendermaßen aussehen:

Ergebnis:

	ORT
1	Hamburg
2	Hamburg
3	Hamburg
4	Kiel
5	Kiel
6	Flensburg
7	Lübeck

Es wird für alle Datensätze in der Tabelle der Ort ausgegeben. Durch das Verwenden des Schlüsselwortes ***DISTINCT*** ändert sich jetzt das Ergebnis wie folgt:

```
SELECT DISTINCT ORT
FROM TBL_KUNDE
```

Ergebnis:

	ORT
1	Hamburg
2	Kiel
3	Flensburg
4	Lübeck

Man erkennt, dass durch DISTINCT Duplikate ausgeschlossen werden, so dass nur noch die verschiedenen Orte jeweils genau einmal angezeigt werden. Dabei werden immer alle Spalten betrachtet, d.h. wenn man DISTINCT auf ORT und GESCHLECHT anwendet, werden die verschiedenen Kombinationen aus beiden Spalten jeweils genau einmal angezeigt. Duplikate über die Kombinationen der verschiedenen Spaltenwerte werden somit entfernt.

```
-- Distinkte Kombinationen selektieren

SELECT DISTINCT ORT, GESCHLECHT
FROM TBL_KUNDE
```

Ergebnis:

	ORT	GESCHLECHT
1	Hamburg	M
2	Hamburg	F
3	Kiel	m
4	Kiel	f
5	Lübeck	M
6	Flensburg	m

Hier sind jetzt jeweils 2 Zeilen für Hamburg und Kiel, da jetzt zwei Spalten im Select sind und somit nur Duplikate über ORT und GESCHLECHT eliminiert werden.

Mittels einem doppelten Minus '--' können ***Kommentare*** in die SQL Statements eingefügt werden.

Zeilen selektieren (WHERE)

Der nächste Bestandteil eines SQL-Statements kann die WHERE Klausel sein. Durch diese ist es möglich, sog. Filter zu definieren, wodurch man nur bestimmte Zeilen selektieren kann:

```
SELECT KUNDE_NR, VORNAME, NACHNAME, ORT
FROM TBL_KUNDE
WHERE ORT='Hamburg'
```

Ergebnis:

	KUNDE_NR	VORNAME	NACHNAME	ORT
1	1	Horst	Huber	Hamburg
2	2	Erika	Schmidt	Hamburg
3	3	Bert	Müller	Hamburg

Es werden dadurch alle Zeilen zurückgegeben mit Ort "Hamburg". In diesem Fall sind das die ersten drei Kunden. Es gibt diverse Operatoren, um Filter zu definieren:

Operator	**Funktion**
=	Gleichheit
<>	Ungleichheit
>	Größer als
<	Kleiner als
>=	Größer oder Gleich
<=	Kleiner oder Gleich
IS (NOT) NULL	Prüft, ob eine Spalte (nicht) NULL ist.
BETWEEN <Wert1> AND <Wert2>	Zwischen <Wert1> und <Wert2>
IN (<wert1>, <wert2>, …)	Enthalten in (…)
LIKE '…..'	Textuelle Gleichheit. Es kann % als Platzhalter für mehrere beliebige Zeichen verwendet werden. _ dient als Platzhalter für genau ein beliebiges Zeichen.
<Operator> ANY (<wert1>,…)	<Operator> kann sein: =, <, >, <>, <=, >= Wird von Oracle aufglöst in: <Operator> <wert1> OR <Operator> <wert2> OR … Wird in der Praxis meist durch andere Vergleiche ersetzt und somit recht ungebräuchlich.
<Operator> ALL (<wert1>,…)	<Operator> kann sein: =, <, >, <>, <=, >= Wird von Oracle aufglöst in: <Operator> <wert1> AND <Operator> <wert2> AND …

	Wird in der Praxis meist durch andere Vergleiche ersetzt und somit recht ungebräuchlich.

Tab.2: SQL Vergleichs Operatoren

Um sich z.B. alle Kunden anzeigen zu lassen, deren Nachnamen mit M beginnen, könnte man folgendes SQL Statement schreiben:

```
SELECT KUNDE_NR, VORNAME, NACHNAME
FROM TBL_KUNDE
WHERE NACHNAME LIKE 'M%'
```

Ergebnis:

KUNDE_NR	VORNAME	NACHNAME
1	3	Bert	Müller
2	4	Hubertus	Meyer-Huber
3	6	Tobias	Maier

Der Filterausdruck 'M%' bedeutet, dass der erste Buchstabe der Zeichenkette ein M sein muss. Die Buchstaben danach sind egal. Das wird durch das % Zeichen erreicht. Es steht als Platzhalter für beliebig viele Zeichen. Als Platzhalter für genau ein Zeichen kann der Unterstrich _ verwendet werden.

Übung 6: Zeigen Sie alle Produkte an, mit einem Preis größer 10€

Übung 7: Zeigen Sie alle Kunden mit Vornamen, Nachnamen und PLZ an, die aus den PLZ Gebieten 24 + 25 kommen.

Berechnungen und Funktionen verwenden

Mit den Spalten im SELECT Abschnitt kann man auch rechnen, z.B. folgendermaßen:

```
SELECT PRODUKT_NR, BEZEICHNUNG, PREIS, PREIS/1.19 AS NETTO, PREIS/1.19*0.19 AS MWST
FROM TBL_PRODUKT
WHERE PREIS/1.19*0.19 > 2
```

Ergebnis:

PRODUKT_NR	BEZEICHNUNG	PREIS	NETTO	MWST
1	1	Rumpsteak	20,95	17,6054…	3,344…
2	2	Grillteller	14,95	12,5630…	2,386…
~~3~~	~~3~~	~~Pizza Salami~~	~~5,60~~	~~4,70…~~	~~0,894…~~

Im obigen Beispiel fällt der dritte Datensatz der Produkttabelle raus, weil die MwSt deutlich unter 2€ liegt. Deshalb ist dieser rot und durchgestrichen dargestellt.

Wie man sieht, können die ganz normalen und bekannten Rechenoperationen auch im SQL angewandt werden. Sowohl im Select als auch in der WHERE Klausel. Im obigen Beispiel führt das dazu, dass nur die Zeilen angezeigt werden, mit einer MwSt von mehr als 2 Euro. Durch AS können Spalten anders benannt werden als sie in der Tabellendefinition heißen. Es können auch Klammern verwendet werden.

Folgende Berechnungsoperatoren stehen zur Verfügung:

Operator	Funktion
+, -, *, /	Addition, Subtraktion, Multiplikation, Division mit Nachkommastellen.
mod (x,y)	Modulo Division
^	Potenz

Tab.3: Berechnungsoperatoren

Es gibt diverse Funktionen, die in den Datenbanken implementiert sind, z.B. zum Runden, Ersetzen von Zeichen in Zeichenketten, Typumwandlung, Rechnen mit Datumswerten, etc. Ich möchte im Folgenden die wichtigsten vorstellen:

Funktion/Syntax	Beschreibung
To_Date(<Wert> [, <Format>])	Wandelt eine Zeichenkette in ein Datum um. Dabei kann eine Formatstring angegeben werden, um die Struktur des Wertes anzugeben. Z.B. Zeichenkette ‘20120201‘ soll in ein Datum umgewandelt werden, dann muss Formatstring ‘YYYYMMDD‘ mit angegeben werden.
To_Char(<Wert> [, <Format>])	Wandelt ein Datum in eine Zeichenkette um. Mittels Formatstring kann definiert werden, wie die Zeichenkette aussehen soll. Z.B. kann ein Datum 01.02.2012 mittels Formatstring ‘YYYYMM‘ leicht in die Form ‘201202‘ gebracht werden.
To_Number (<Zeichenketten>)	Wandelt eine Zeichenkette in eine Zahl um.
Round (<Zahl>, <Stellen>)	Rundet die Zahl <Zahl> auf <Stellen> Stellen.
Substr (<Text>, <Start>, <Anz. Zeichen>)	Liefert einen Ausschnitt aus <Text> von Position <Start> mit <Anz. Zeichen> Länge
Length (<Text>)	Liefert die Länger der Zeichenkette <Text> zurück.
InStr (<Text>, <Zeichen>, <Start Pos.>)	Sucht <Zeichen> in <Text> mit Start an <Start Pos.> und liefer die Position zurück.

Replace (<Text>, <Zeichen>, <Zeichen_Neu>)	Ersetzt alle Zeichen <Zeichen> durch <Zeichen_Neu> in <Text>.
Concat (<Text1>, <Text2>, …)	Verbindet die Texte 1 … n zu einer Zeichenketten. Alternativ kann auch der Operator || verwendet werden.
LTrim/RTrim (<Text> [, <Zeichen>])	Schneidet alle Zeichen <Zeichen> links bzw. rechts von <Text> ab. Wenn kein Zeichen angegeben wurde, werden Leerzeichen entfernt.
NVL (<Feld>, <Nullwertzeichen>)	Ersetzt NULL Werte in <Feld> durch <Nullwertzeichen>.
ADD_MONTHS (<Datum>, <Monate<)	Zählt <Monate> Monate auf <Datum> drauf und liefert das entsprechende Datum zurück.
LAST_DAY (<Datum>)	Liefert den letzten Tag des Monats aus <Datum>.
UPPER / LOWER (<Text>)	Wandelt alle Zeichen von <Text> in Groß- bzw. Kleinbuchstaben um.
LPad/RPad(<Text>, <Breite> [, <Zeichen>])	Füllt den String <Text> auf bis zu <Breite> Zeichen auf mit dem Zeichen <Zeichen>. Wenn kein Zeichen angegeben wurde, wird der String mit Leerzeichen aufgefüllt.
ABS (<Zahl>)	Liefert die absolute Zahl zurück.
SYSDATE / SYSTIMESTAMP	Liefert das akt. Systemdatum zurück bzw. den akt. Systemzeitstempel (d.h. Datum + Uhrzeit)
TRUNC (<Zahl>, <Anzahl>)	Schneidet die <Zahl> ab bis auf <Anzahl> Nachkommastellen. Es wird nicht gerundet. Lässt man <Anzahl> weg, wird bis zum Komma abgeschnitten.

Tab.4: Oracle Standardfunktionen

Unter
http://docs.oracle.com/cd/B28359_01/olap.111/b28126/dml_commands_1029.htm#OLADM780
findet man eine Übersicht mit allen möglichen Formatstring-Bestandteilen.

Funktionen können sowohl im SELECT- als auch im WHERE-Teil eine SQL Statements verwendet werden, woraus noch weitergehende Filtermöglichkeiten entstehen:

```
SELECT KUNDE_NR, VORNAME, NACHNAME, GEBURTSTAG
FROM TBL_KUNDE
WHERE to_number(  to_char(GEBURTSTAG, 'YYYY'))>=1980
```

Ergebnis:

	KUNDE_NR	VORNAME	NACHNAME	GEBURTSTAG
1	2	Erika	Schmidt	05.10.1985
2	6	Tobias	Maier	03.03.1992

Wie man sieht, können auch mehrere Funktionen miteinander verschachtelt werden. Dieses SQL selektiert alle Kunden, die 1980 oder später geboren wurden. Mittels String-Funktion können alle männlichen Kunden gewählt werden:

```
SELECT VORNAME, NACHNAME, GESCHLECHT
FROM TBL_KUNDE
WHERE UPPER(GESCHLECHT)='M'
```

Ergebnis:

	VORNAME	NACHNAME	GESCHLECHT
1	Horst	Huber	M
2	Bert	Müller	m
3	Hubertus	Meyer-Huber	M
4	Tobias	Maier	M
5	Fabian	Lindemann	m

Wieso verwendet man die UPPER-Funktion? Es kann ja sein, dass mal ein großes und mal ein kleines M für das Geschlecht eingegeben wurde. Alternativ hätte man das Ganze auch mit Hilfe des IN Operators erreichen können: IN('M', 'm')

Übung 8: Zeigen Sie alle Kunden an, deren Vornamen mit F beginnen und verwenden Sie dabei Funktionen (und nicht den LIKE Operator)!

Übung 9: Selektieren Sie alle Kunden, deren Nachnamen auf 'mann' enden und verwenden Sie dabei ebenfalls Funktionen (und nicht den LIKE Operator)!

Übung 10: Selektieren Sie alle Kunden, die im 1. Quartal eines Jahres Geburtstag haben

Mehrere Filter verknüpfen

Oftmals ist es notwendig, Datensätze anhand mehrere Kriterien zu selektieren. Z.B. könnte es interessant sein, alle Kunden zu anzuzeigen, die nach 1970 geboren sind und weiblich sind:

```
SELECT KUNDE_NR, VORNAME, NACHNAME, GESCHLECHT, GEBURTSTAG
FROM TBL_KUNDE
WHERE to_number( to_char(GEBURTSTAG, 'YYYY'))>=1970
AND GESCHLECHT IN('F', 'f')
```

	KUNDE_NR	VORNAME	NACHNAME	GESCHLECHT	GEBURTSTAG

1	2	Erika	Schmidt	F	05.10.1985
~~5~~	~~Hanna~~	~~von Bergmann~~	~~f~~	~~17.09.1965~~	

Man kann Filter mittels ***AND*** miteinander verknüpfen. Damit müssen beide Bedingungen erfüllt sein, damit der Datensatz zurückgeliefert wird. Der zweite Datensatz wird in diesem Fall nicht angezeigt, da der Geburtstag nicht passt.

Alternativ können Bedingungen auch mit ***OR*** kombiniert werden. Dann muss entweder Bedingung A oder B zutreffen:

```
SELECT VORNAME, NACHNAME, GESCHLECHT, GEBURTSTAG
FROM TBL_KUNDE
WHERE GESCHLECHT IN('F', 'f')
OR  to_number( to_char(GEBURTSTAG, 'YYYY'))>=1970
```

Ergebnis:

KUNDE_NR	VORNAME	NACHNAME	GESCHLECHT	GEBURTSTAG
1	2	Erika	Schmidt	F	05.10.1985
2	3	Bert	Müller	M	03.02.1979
3	5	Hanna	von Bergmann	f	17.09.1965
4	6	Tobias	Maier	M	03.03.1992
5	7	Fabian	Lindemann	m	01.09.1973

Es werden also alle weibliche Kunden zurückgeliefert oder Kunden, die nach 1970 geboren wurden. Komplex wird das Ganze, wenn man AND und OR miteinander verknüpft. Dabei hat der AND Operator Vorrang vor dem OR Operator:

```
SELECT VORNAME, NACHNAME, ORT, GESCHLECHT
FROM TBL_KUNDE
WHERE ORT='Hamburg' OR ORT='Kiel' AND GESCHLECHT IN('M', 'm')
```

Ergebnis:

KUNDE_NR	VORNAME	NACHNAME	ORT	GESCHLECHT
1	1	Horst	Huber	Hamburg	M
2	2	Erika	Schmidt	Hamburg	F
3	3	Bert	Müller	Hamburg	M
4	4	Hubertus	Meyer-Huber	Kiel	m

Je nachdem, was man nun auswerten möchte, muss man ggf. Klammern setzen. So wie das Statement oben definiert wurde, werden entweder Kunden aus Hamburg zurück geliefert (egal ob männlich oder weiblich) oder männliche Kunden aus Kiel.

Wenn man möchte, dass männliche Kunden angezeigt werden, die aus Kiel oder Hamburg kommen, muss man das SQL folgendermaßen erweitern:

```
SELECT VORNAME, NACHNAME, ORT, GESCHLECHT
FROM TBL_KUNDE
WHERE (ORT='Hamburg' OR ORT='Kiel') AND GESCHLECHT IN('M', 'm')
```

Ergebnis:

	KUNDE_NR	VORNAME	NACHNAME	ORT	GESCHLECHT
1	1	Horst	Huber	Hamburg	M
2	3	Bert	Müller	Hamburg	M
3	4	Hubertus	Meyer-Huber	Kiel	m

Der Datensatz „Erika Schmidt“ fällt jetzt auch noch raus, da die zurückgelieferten Datensätze folgende Bedingungen erfüllen müssen:

1. männlich
2. Aus Kiel oder Hamburg

Das wurde erreicht durch Verwenden von Klammern. Es wird also zunächst die Bedingung (ORT='Hamburg' OR ORT='Kiel') ausgewertet. In der Ergebnismenge davon wir dann geguckt, welche Kunden männlich sind.

Im Zusammenhang mit AND und OR gibt es noch den Operator NOT. Es kann damit z.B. geprüft werden, welche Kunden nicht aus Kiel, Flensburg oder Lübeck kommen:

```
SELECT VORNAME, NACHNAME, ORT
FROM TBL_KUNDE
WHERE ORT NOT IN ('Hamburg', 'Flensburg', 'Lübeck')
```

Ergebnis:

	KUNDE_NR	VORNAME	NACHNAME	ORT
1	4	Hubertus	Meyer-Huber	Kiel
2	5	Hanna	von Bergmann	Kiel

Übung 11: Selektieren Sie alle Produkte mit der Produktgruppe 1 und einem Preis größer 15€

Übung 12: Selektieren Sie alle Produkte mit einer MwST Betrag< 0,75€ oder >2€ aus den Produktgruppen 1,2 oder 4. Zeigen Sie dabei alle Spalten der Produkt Tabelle an und zusätzlich noch den Betrag der MwSt. (MwSt-Satz: 19%)

Übung 13: Selektieren Sie alle Bestellungen (Bestell-Nr), die min 2 Stück der Produkte 1…5 oder min. 3 Stück der Produkte 6 … 15 beinhalten.

Ergebnisse sortieren

Man kann die zurückgelieferten Datensätze auch sortieren, z.B. nach PLZ:

```
SELECT KUNDE_NR, VORNAME, NACHNAME, PLZ
FROM TBL_KUNDE
ORDER BY PLZ ASC
```

Ergebnis:

	KUNDE_NR	VORNAME	NACHNAME	PLZ
1	1	Horst	Huber	20357
2	3	Bert	Müller	22123
3	2	Erika	Schmidt	22512
4	7	Fabian	Lindemann	23809
5	4	Hubertus	Meyer-Huber	24106
6	5	Hanna	von Bergmann	24145
7	6	Tobias	Maier	26105

Ein anderes Beispiel zum Sortieren der Ergebnisse:

```
SELECT KUNDE_NR, VORNAME, NACHNAME, ORT
FROM TBL_KUNDE
ORDER BY 4 DESC, 1 ASC
```

Ergebnis:

	KUNDE_NR	VORNAME	NACHNAME	ORT
1	7	Fabian	Lindemann	Lübeck
2	4	Hubertus	Meyer-Huber	Kiel
3	5	Hanna	von Bergmann	Kiel
4	1	Horst	Huber	Hamburg
5	3	Bert	Müller	Hamburg
6	2	Erika	Schmidt	Hamburg
7	6	Tobias	Maier	Flensburg

Wie man sieht kann man durch ***ASC*** oder ***DESC*** angeben, in welche Richtung sortiert werden soll. Sortierspalten können entweder per Namen oder als Position angegeben werden. Mehrere Sortierkriterien können durch Kommata getrennt werden.

Mittels ***NULLS FIRST*** und ***NULLS LAST*** kann bei der Sortierung noch angegeben werden, dass Nullwerte am Anfang oder ans Ende der Liste gestellt werden. Der Rest wird dann sortiert wie beschrieben.

Zum Abschluss dieses Kapitels noch ein kleineres Thema: In Oracle existiert eine Tabelle, die DUAL heißt. In dieser befindet sich genau ein Datensatz. Diese kleine Tabelle kann manchmal hilfreich sein, weil man bei Oracle im SQL immer eine Tabelle in der FROM Klausel angeben muss. Wenn ich z.B. nur mal kurz eine Funktion ausprobieren möchte kann ich das wie folgt machen:

```
SELECT SYSDATE
```

```
FROM dual;
```

Man bekommt dann genau einen Wert zurückgeliefert, nämlich das Systemdatum in diesem Fall.

Aufgaben

1) Stellen Sie alle Kunden dar, die älter als 40 Jahre sind. Zeigen Sie auch das jeweilige Alter mit an.

2) Welcher Kunde hat die meisten Bestellung im 2. Quartal 2013 aufgegeben und wie viele?

3) Ihre Fahrerin Liese ist immer zu langsam beim Kunden. Insofern sind die meisten Pizzen kalt und es gab viele Beschwerden dieser Kunden. Um allen Kunden, die von Liese beliefert wurden, ein kostenloses Tiramisu als Widergutmachung anzubieten, benötigen Sie eine Liste mit allen Bestellungen des 1. Quartals 2013, die Liese beliefert hat. Keine Selbstabholer.

4) Erstellen Sie einen Bericht, der die Bestellungen für die Hamburger Kunden im 1. Quartal 2013 zeigt.

5) Welches Produkt war das unbeliebteste bei den Kunden im März 2013?

6) Welche Kunden wurden im Februar oder August 2013 von Mitarbeiter Emil Iversen und von Fahrer Liese Müller beliefert oder von Mitarbeiter Emil Iversen und von Fahrer Peter Peters beliefert? Verwenden sie nur AND,OR, NOT Operatoren.