Vorbemerkungen

Bisher haben wir Datenbanken nur über einzelne Tabellen kennen gelernt. Stehen mehrere Tabellen in gewissen Beziehungen zur Beschreibung einer bestimmten Situation, so spricht man von einer **relationalen Datenbank** (zum relationalen Datenbankmodell folgen später mehr Informationen).
Damit Datenbanken geeignet benutzt werden können, muss dafür eine betriebssystem- und plattformunabhängige Möglichkeit zur Abfrage, Erstellung und Manipulation vorhanden sein, dies insbesondere im Hinblick auf die Erfordernisse des Internets, wo *Datenbankserver* die vielfältigsten Anfragen zu Informationen aus Datenbanken verarbeiten müssen.
Als Quasi-Standard hat sich als Abfragesprache **SQL** (Structured Query Language) mittlerweile durchgesetzt. Die Ursprünge gehen auf ein IBM-Forschungsprojekt zurück. 1986 wurde SQL als Standard von der amerikanischen Normbehörde (ANSI) übernommen, 1987 folgte die internationale Normierung (ISO). SQL ist die Sprache, mit der die meisten relationalen Datenbanken erstellt, manipuliert und abgefragt werden.
SQL lässt sich innerhalb der Programmiersprachen als Sprache der 4. Generation bezeichnen (4GL), denn sie ist im Gegensatz zu prozeduralen Sprachen der 3. Generation, die durch Datenstrukturen und Steuerstrukturen gekennzeichnet sind, nicht-prozedural und mengenorientiert. Nicht-prozedural heißt hier, dass der Fragesteller eine Frage stellt, aber keinen Algorithmus zur Lösung vorgeben muss. Es ist nicht nötig, sich aus Programmierersicht Gedanken über das Öffnen und Schließen einer Datei und das schrittweise Durchgehen geeigneter Datensätze zu machen.
SQL wird auch von *LibreOffice-Base* unterstützt. Im Folgenden werden die schon behandelten Abfragen aus SQL-Sicht dargestellt und ihre Umsetzung mit *Base* beschrieben.

Auswahl-Abfragen mit SELECT

Mit Auswahl-Abfragen sind insbesondere die Operationen *Projektion* und *Selektion* auf einer Tabelle von Interesse (Die Abfrage über mehrere Tabellen hinweg folgt später).
Die vereinfachte Syntax des `SELECT`-Befehls lautet wie folgt:

```
SELECT [DISTINCT | ALL] select_expression,... FROM tables ...
[WHERE where_definition]
[GROUP BY feld,...]
[ORDER BY feld [ASC | DESC] ,...
[LIMIT [offset,] rows]  ;
```

Am einfachsten versteht man das an Beispielen. Im Folgenden sei die Tabelle *Mitarbeiter* die Grundlage der Beispiele.

MitarbNr	VorgesetztenNr	AbtNr	Name	GebDatum	Telephon
1	NULL	3	Christoph Reeg	13.05.1979	NULL
2	1	1	junetz.de	05.03.1998	069-764758
3	1	1	Uli	NULL	NULL
4	3	1	JCP	NULL	069-764758
5	1	2	Maier	NULL	06196-671797
6	5	2	Meier	NULL	069-97640232

Selektion

1. Die kürzestmögliche `SELECT`-Anweisung ist `SELECT * FROM Mitarbeiter;`

   Es sollen alle Mitarbeiter ausgegeben werden, d.h. es wird dann einfach die Tabelle wie oben dargestellt geliefert.

2. `SELECT * FROM Mitarbeiter WHERE Name="Maier";` liefert

MitarbNr	VorgesetztenNr	AbtNr	Name	GebDatum	Telephon
5	1	2	Maier	NULL	06196-671797

3. ```
   SELECT * FROM Mitarbeiter
   WHERE (VorgesetztenNr=1) AND (AbtNr=1);
   ```

   liefert

MitarbNr	VorgesetztenNr	AbtNr	Name	GebDatum	Telephon
2	1	1	junetz.de	05.03.1998	069-764758
3	1	1	Uli	NULL	NULL

Mehr zu den Bedingungen für `WHERE` folgt unten im Abschnitt `where_definition`.

Projektion

1. `SELECT Telephon FROM Mitarbeiter;` oder

   `SELECT ALL Telephon FROM Mitarbeiter;` liefert

Telephon
NULL
069-764758
NULL
069-764758
06196-671797
069-97640232

2. `SELECT DISTINCT Telephon FROM Mitarbeiter;` liefert
   ```

Telephon
NULL
069-764758
06196-671797
069-97640232

Mehrfach vorkommende Einträge werden nur einmal angezeigt.

3. `SELECT AbtNr, Name, Telephon FROM Mitarbeiter;`

   liefert

AbtNr	Name	Telephon
3	Christoph Reeg	NULL
1	junetz.de	069-764758
1	Uli	NULL
1	JCP	NULL
2	Maier	06196-671797
2	Meier	NULL

Sortierung

Mit `ORDER BY` wird festgelegt, nach welcher Spalte bzw. welchen Spalten sortiert werden soll. Mit `ASC` wird aufsteigend, mit `DESC` absteigend sortiert. Wenn nichts angegeben wird, wird aufsteigend sortiert.

1. `SELECT * FROM Mitarbeiter ORDER BY Name;` oder

   `SELECT * FROM Mitarbeiter ORDER BY Name ASC;` liefert

MitarbNr	VorgesetztenNr	AbtNr	Name	GebDatum	Telephon
1	NULL	3	Christoph Reeg	13.05.1979	NULL
4	3	1	JCP	NULL	069-764758
2	1	1	junetz.de	05.03.1998	069-764758
5	1	2	Maier	NULL	06196-671797
6	5	2	Meier	NULL	069-97640232
3	1	1	Uli	NULL	NULL

2. `SELECT * FROM Mitarbeiter ORDER BY Name DESC;` liefert

MitarbNr	VorgesetztenNr	AbtNr	Name	GebDatum	Telephon
3	1	1	Uli	NULL	NULL
6	5	2	Meier	NULL	069-97640232
5	1	2	Maier	NULL	06196-671797
2	1	1	junetz.de	05.03.1998	069-764758
4	3	1	JCP	NULL	069-764758
1	NULL	3	Christoph Reeg	13.05.1979	NULL

3. `SELECT * FROM Mitarbeiter ORDER BY GebDatum,Name;` liefert

MitarbNr	VorgesetztenNr	AbtNr	Name	GebDatum	Telephon
4	3	1	JCP	NULL	069-764758
5	1	2	Maier	NULL	06196-671797
6	5	2	Meier	NULL	069-97640232
3	1	1	Uli	NULL	NULL
1	NULL	3	Christoph Reeg	13.05.1979	NULL
2	1	1	junetz.de	05.03.1998	069-764758

   Da die ersten vier Mitarbeiter kein Geburtsdatum haben, wird ein zweites Sortierkriterium für *Name* eingegeben.

Funktionen

1. `SELECT count(*) FROM Mitarbeiter;` liefert

count(*)
6

2. Mit `GROUP BY` können in Verbindung mit *Gruppenfunktionen* Gruppen gebildet werden.

   `SELECT count(*), AbtNr FROM Mitarbeiter GROUP BY AbtNr;` liefert

count(*)	AbtNr
3	1
2	2
1	3

Näheres zu Funktionen folgt bei Bedarf.

LIMIT

Mit `LIMIT [offset,] rows` kann angegeben werden, wie viele Zeilen maximal von der `SELECT`-Anweisung zurück geliefert werden sollen. Mit `offset` kann angegeben werden, ab welcher Zeile angefangen werden soll. Man denke dabei an die Ergebnisdarstellung einer Internet-Suchmaschinenabfrage.
`SELECT * FROM Tabelle LIMIT 5,10;` liefert die Zeilen 6 bis 15 der Tabelle.
`SELECT * FROM Tabelle LIMIT 5;` liefert die ersten fünf Zeilen der Tabelle.
`SELECT * FROM Tabelle LIMIT 0,5;` liefert ebenso die ersten fünf Zeilen.

where_definition

Vergleichsoperatoren

Teilbedingungen können mit `AND`, `OR` und `NOT` zu Bedingungen zusammen gesetzt werden. An Vergleichsoperatoren sind in SQL folgende verfügbar:

Operator	**Bedeutung**
`=`	gleich
`<>` oder `!=`	ungleich
`>`	größer
`<`	kleiner
`>=`	größer gleich
`<=`	kleiner gleich
`IS`	gleich (für Vergleich mit `NULL`)

```
SELECT Name, AbtNr FROM Mitarbeiter
WHERE (AbtNr=1) OR (VorgesetztenNr IS NULL);
```

liefert

Name	AbtNr	VorgesetztenNr
Christoph Reeg	3	NULL
junetz.de	1	1
Uli	1	1
JCP	1	3

LIKE

```
SELECT Name, Telephon FROM Mitarbeiter
WHERE (Telephon LIKE "%96-%");
```

liefert

Name	Telephon
Maier	06196-671797

```
SELECT Name, Telephon FROM Mitarbeiter
WHERE (Name LIKE "M_ier");
```

liefert

Name	Telephon
Maier	06196-671797
Meier	069-97640232

`LIKE` hat seine Entsprechung im `WIE` in *Base*.

Zeichen in SQL	Zeichen in *Base*	Bedeutung
`%`	`*`	steht für eine bel. Anzahl (auch 0) von Zeichen
`_`	`?`	steht für genau ein bel. Zeichen

BETWEEN

`BETWEEN` wählt alle Werte aus, die zwischen dem unteren und oberen Wert (einschließlich) liegen.
`SELECT * FROM Mitarbeiter WHERE (AbtNr BETWEEN 2 AND 3);` liefert

MitarbNr	VorgesetztenNr	AbtNr	Name	GebDatum	Telephon
1	NULL	3	Christoph Reeg	13.05.1979	NULL
5	1	2	Maier	NULL	06196-671797
6	5	2	Meier	NULL	069-97640232

In *Base* lautet die Entsprechung `Zwischen...und...`.

IN

Wie in *Base* benutzt man `IN`.

```
SELECT * FROM Mitarbeiter
WHERE Telephon IN ("06196-671797","069-97640232");
```

liefert

MitarbNr	VorgesetztenNr	AbtNr	Name	GebDatum	Telephon
5	1	2	Maier	NULL	06196-671797
6	5	2	Meier	NULL	069-97640232

SQL und Base

In der Entwurfsansicht einer Abfrage kann man SQL wählen und bekommt die SQL-Ansicht dargestellt. Ebenso ist das direkte Eingeben von SQL-Befehlen möglich.

Übungen

1. Wiederhole einige Abfragen des Abfrage-Arbeitsblattes und lasse dir die entsprechenden SQL-Befehle anzeigen. Notiere sie dir.

2. Wiederhole die Aufgaben zum Filtern von Datensätzen durch direkte Eingabe der entsprechenden SQL-Befehle.