Structured Query Language (SQL) 1

1. Grundlagen und Hilfsmittel

Structured Query Language, kurz **SQL**, wurde in den 70er Jahren bei IBM entwickelt, als eine Arbeitsgruppe die erste relationale Datenbank programmiert. SQL wird durch fast alle Datenbanksysteme mehr oder weniger gut unterstützt. Für SQL sind Standards definiert. Dennoch gibt es Unterschiede bei der für die wichtigsten relationalen Datenbank-Managementsysteme jeweils verwendeten SQL. Dies hat zwei Gründe:

1) der SQL-Standard ist relativ komplex und eine Implementierung des kompletten Standards somit oft nicht praxisgerecht und
2) jeder Datenbankanbieter möchte sein Produkt von den Mitbewerbern abgrenzen.

Damit Sie für die folgenden SQL-Beispiele nicht jedes Mal ein neues Programm schreiben müssen, finden Sie im Verzeichnis SQL das Programm SQLEDIT sowie die folgende Datenbank:

Artikel (Nr, Name, Einkaufspreis, Verkaufspreis, Bestand)
Kunden (Nr, Name, Strasse, Ort, PLZ, Telefon)
Rechnungen (Nr, KundenNr, Datum, Gesamtnetto, Bemerkung)
Rechnungsdaten (RechnungsNr, ArtikelNr, ArtikelZahl)

Das Formular des Programms enthält in der Mitte ein Datenbank-Grid und unten ein Memo-Feld, in das die SQL-Befehl eingetragen werden können.
Oben befindet sich eine kleine Werkzeugleiste:

- der erste Symbol öffnet eine Auswahlbox, mit der der Pfad eingetragen werden kann
- mit dem Blitz können Sie die eingetragenen Befehle ausführen
- das dritte Symbol öffnet die Database-Info. Bei Klick auf eine Tabelle werden die zugehörigen Felder angezeigt.
- die Tür dient zum Beenden des Programms.

Hinweis:
Die SQL-Befehlswörter werden sowohl im Text als auch in den Beispielen grundsätzlich groß geschrieben. Tabellennamen und Feldbezeichner erscheinen in Kleinbuchstaben, wobei einzelne Großbuchstaben die Lesbarkeit erhöhen.

Zentralabitur Niedersachsen
Im Zentralabitur in Niedersachsen wird den Aufgaben zum Thema Datenbanken folgende SQL-Referenz als Hilfestellung beigefügt:

SELECT-Anweisung

```
SELECT [ALL|DISTINCT] SpaltenName {, SpaltenName}
   FROM TabellenName {, TabellenName}
   [WHERE Suchbedingung]
   [GROUP BY SpaltenName {, SpaltenName}]
   [HAVING  Suchbedingung ]
   [ORDER BY SpaltenName [ASC|DESC] {, SpaltenName [ASC|DESC]}]
```

Operatoren:
+, -, *, /, =; != (ungleich), >, <, >=, <=, AND, OR, NOT

Arithmetische Gruppenfunktionen:
AVG ( ), COUNT ( ), MAX ( ), MIN ( ), SUM ( )

2. Daten abfragen mit der SELECT-Anweisung

Den SELECT-Befehl finden Sie in fast jeder SQL-Anweisung. Mit SELECT wählen Sie Datenfelder, die bestimmten Bedingungen genügen, aus einer oder auch mehreren Tabellen aus.
Syntax:

```
SELECT <Liste der Spalten> [AS] <Aliasname> FROM <Tabellenliste>
```

Wenn Sie alle Spalten einer Tabelle selektieren möchten, können Sie statt der Aufzählung der Namen auch einen Stern (*) verwenden.

a) Die einfachste Form einer SELECT-Anweisung

```
SELECT * FROM Artikel
```

... listet alle Datensätze aus der Tabelle „Artikel“ auf:

b) Mit SELECT Name, Einkaufspreis FROM artikel
... wählen Sie nur die Spalten „Name“ und „Einkaufspreis“ aus der Tabelle „Artikel“:

c) Ein Problem (z. B. bei der Auswahl von Name und Vorname aus einer großen Importdatei) sind „doppelte“ Datensätze. Um diese Redundanz zu vermeiden, verwenden Sie den optionalen Parameter DISTINCT. Die Datensätze verbleiben aber auf jeden Fall in der Tabelle.
Beispiel:

```
SELECT DISTINCT Name
FROM Kunden
```

... liefert eine Tabelle, die jeden Kundennamen nur einmal aufführt.
Das Gegenteil von DISTINCT erreichen Sie mit dem Parameter ALL, der standardmäßig voreingestellt ist.

d) Mit einer AS-Anweisung können Sie den einzelnen Tabellenspalten neue Bezeichner zuweisen.

```
SELECT Name AS Artikelbezeichnung, Verkaufspreis AS Preis
FROM Artikel
```

3. Verbindung von Tabellen mit JOIN

Im Zuge der Normalisierung von Datenbanken werden die Daten in verschiedene Tabellen aufgeteilt, um eine redundanzfreie Speicherung zu ermöglichen und Einfüge-, Lösch- und Änderungsanomalien zu vermeiden. Für die praktische Anwendung muss man aber Einträge aus verschiedenen Tabellen verknüpfen. Dazu erstellt man mittels der JOIN-Verknüpfung eine neue virtuelle Tabelle, die alle notwendigen Daten enthält.
Grundsätzlich kann man mit dem JOIN zwei beliebige Tabellen miteinander verknüpfen. Das Ergebnis ist eine Tabelle, die alle möglichen Kombinationen von Datensätzen aus Tabelle 1 und aus Tabelle 2 enthält. Meistens werden Tabellen mit JOIN verbunden, wenn sie etwas Gemeinsames haben, z.B. einen Fremdschlüssel.

a) Der Befehl

```
SELECT Name, ArtikelZahl, Einkaufspreis
FROM Artikel
RIGHT JOIN Rechnungsdaten
ON Artikel.Nr = Rechnungsdaten.ArtikelNr
```

liefert eine Tabelle, die Namen, Artikelanzahl und Einkaufspreis umfasst, wobei der Parameter ON dafür sorgt, dass nur die Kombinationen aus den beiden Tabellen gebildet werden, die übereinstimmende Artikelnummer haben. Alternativ wird mit gleicher Bedeutung (z.B. in den niedersächsischen Abituraufgaben) auch folgende Schreibweise benutzt

```
SELECT Name, ArtikelZahl, Einkaufspreis
FROM Artikel, Rechnungsdaten
WHERE Artikel.Nr = Rechnungsdaten.ArtikelNr
```

Hierbei sorgt die WHERE-Klausel für die Auswahl der „richtigen" Kombinationen, die in der Artikelnummer übereinstimmen.

Hinweis: Bei einem RIGHT JOIN werden auch diejenigen Elemente der zweiten Tabelle aufgeführt, für die keine entsprechenden Einträge in der ersten Tabelle vorhanden sind. Die leeren Spalten werden dann mit NULL aufgefüllt, dem Schlüsselwort für leere Felder.

b) Verwenden Sie mehrere Tabellen mit gleichen Spaltennamen in einer SELECT-Anweisung, müssen Sie in der Liste der Feldbezeichner zusätzlich den Tabellennamen angeben:

```
SELECT Kunden.Nr, Name, Rechnungen.Nr
FROM Kunden, Rechnungen
WHERE Kunden.Nr = Rechnungen.KundenNr
```

c) Sind viele Feldern zu selektieren, ist es recht mühsam, jedesmal den vollen Tabellennamen in die SQL-Anweisung einzusetzen. Zu diesem Zweck können Sie Alias-Namen verwenden. Die folgende SQL-Anweisung entspricht der obigen:

```
SELECT K.Nr, K.Name, R.Nr
FROM Kunden K, Rechnungen R
WHERE K.Nr = R.KundenNr
```

d) Werden mehrere Tabellen mit JOIN verknüpft, so ist für jede Verknüpfung eine Bedingung anzugeben, die die Anzeige auf zusammengehörende Datensätze beschränkt:

```
SELECT Kunden.Name, Rechnungsdaten.ArtikelNr
FROM Kunden
RIGHT JOIN Rechnungen
ON Kunden.Nr = Rechungen.KundenNr
RIGHT JOIN Rechnungsdaten
ON Rechnungen.Nr = Rechnungsdaten.Nr
```

Alternative Schreibweise:

```
SELECT Kunden.Name, Rechnungsdaten.ArtikelNr
FROM Kunden, Rechnungen, Rechnungsdaten
WHERE (Kunden.Nr = Rechungen.KundenNr) AND
      (Rechnungen.Nr = Rechnungsdaten.Nr)
```

4. Daten filtern mit Hilfe der WHERE-Klausel

Mit den bisherigen SELECT-Anweisungen konnten Sie immer nur alle Datensätze auf einmal auswählen. Zur Selektion bestimmter Datensätze verwenden Sie – wie im vorigen Abschnitt beim JOIN von Tabellen bereits gezeigt - die WHERE-Klausel in Verbindung mit einem logischen Ausdruck und dem SELECT-Befehl.

a) SELECT * FROM Artikel
WHERE (Bestand < 10)
... listet alle Artikel auf, von denen weniger als 10 Stück vorhanden sind:

b) Für die Auswahl von Elementen, die in einem bestimmten Bereich liegen, stehen Ihnen zwei verschiedene Möglichkeiten zur Verfügung. Beispiel Auswahl aller Artikel, die mehr als 100 Euro und weniger als 600 Euro kosten:
Erstens: Beschreiben des Intervalls mit „<„„>„„>=„„<=„ sowie der AND-Verknüpfung

```
SELECT *
FROM Artikel
WHERE (Verkaufspreis > 100) AND (Verkaufpreis < 600)
```

Zweitens: der BETWEEN-Ausdruck (Grenzen werden nicht eingeschlossen!)

```
SELECT *
FROM Artikel
WHERE Verkaufspreis BETWEEN 100 AND 600
```

c) Mit einer einfachen Negation der obigen Bedingung können Sie alle Artikel auflisten, die weniger als 100 Euro und mehr als 600 Euro kosten. Beachten Sie, daß sowohl 100 als auch 600 mit zum Definitionsbereich gehören und angezeigt werden!

```
SELECT *
FROM Artikel
WHERE Verkaufspreis NOT BETWEEN 100 AND 600
```

d) Für die Suche nach Zeichenketten können Sie mehrere Operatoren verwenden. Sowohl das Gleichheitszeichen (=) als auch der LIKE-Ausdruck liefern dasselbe Ergebnis. Beachten Sie die Groß-/Kleinschreibung!

```
SELECT *
FROM Artikel
WHERE Name = 'VISUAL BASIC 3.0'
```

e) SQL unterstützt verschiedene Formen von Platzhaltern. Für ein beliebiges Zeichen können Sie den Unterstrich (_) verwenden, das Prozentzeichen (%) steht für eine beliebige Anzahl von Zeichen. Beispiel: Ein Kunde sucht einen Artikel, von dem nur bekannt ist, daß er mit „VISUAL“ beginnt. Durch die Verwendung eines Platzhalters nach dem Suchstring können Sie alle Artikel, die dem Kriterium entsprechen, auflisten.

```
SELECT *
FROM Artikel
WHERE Name LIKE 'VISUAL%'
```

f) Eine weitere Möglichkeit, Datensätze zu selektieren, stellt die IN-Klausel dar:

```
SELECT *
FROM Kunden
WHERE Ort IN ('Hamburg', 'München',' Dresden')
```

...sucht uns alle Datensätze heraus, wo Ort in der Liste aufgeführt ist.

g) Neben den bisherigen Möglichkeiten, nach etwas Bestimmten zu suchen, kann man auch nach „nichts“ suchen. „Nichts“, also kein Inhalt, wird in im betreffenden Feld mit dem Schlüsselwort NULL gekennzeichnet. Eine Liste aller Kunden, die kein Telefon haben, erhält man mit :

```
SELECT *
FROM Kunden
WHERE (Telefon IS NULL)
```

4. Berechnungen in SQL-Anweisungen

In vielen Fällen möchte man statt einer Liste eine Summe oder das Maximum ausgeben (z. B. die Anzahl der verkauften Artikel oder das Durchschnittsgehalt in der Firma). SQL unterstützt diesen Wunsch durch eine Reihe von arithmetischen Gruppenfunktionen:

- Summenbildung (SUM)
- Mittelwert (AVG)
- Minimum (MIN)
- Maximum (MAX)
- Zählen (COUNT)

Zu den Gruppenfunktionen SUM, AVG, MIN, MAX und COUNT ist jeweils der Bereich angegeben, auf dem diese Rechenoperation durchgeführt werden soll. Im einfachsten Fall ist dies der Name einer Spalte:

a)
```
SELECT AVG(Verkaufspreis)
FROM Artikel
```
... berechnet den Durchschnittspreis aller Artikel.

b)
```
SELECT SUM (Gesamtnetto) AS Nettojahresumsatz
FROM Rechnungen
WHERE (Datum>=01.01.2007) AND (Datum<=31.12.2007)
```
... berechnet den Nettoumsatz im Jahr 2007 und gibt ihn mit der Bezeichnung „Nettojahresumsatz“ aus.
Hinweis: Beachten Sie bitte, dass das Datum je nach Datenbank bzw. Aufgabenstellung in verschiedenen Formaten gespeichert werden kann. Neben DD.MM.JJJJ kann z.B. auch das Format JJJJ-MM-DD verwendet werden!

c) Neben den arithmetischen Gruppenfunktionen können auch einfache Operatoren (für Zentralabitur Niedersachsen siehe Liste oben) verwendet werden.
```
SELECT Name, Verkaufspreis * 1.1 AS Preis
FROM Artikel
```
... gibt die Liste aller Artikel mit 10% Aufschlag aus. Die Änderung des Preises hat keine Auswirkung auf die Preise in der Tabelle, lediglich die Abfrage enthält diese Werte. Der neue Feldname ist „Preis“.

5. Gruppenbildung mit GROUP BY

Im Zusammenhang mit Berechnungsfunktionen oder mit Listenausgaben tritt häufig das Problem auf, bestimmte Gruppen innerhalb einer Tabelle zu bilden.

a) Sie haben eine Tabelle
```
Personal (Nr, Name, Vorname, AbteilungsNr)
```
Um die Anzahl der Mitarbeiter in den einzelnen Abteilungen zu bestimmen, können Sie zwei verschiedene Möglichkeiten nutzen:
```
SELECT AbteilungsNr, COUNT(abtnr) FROM Personal WHERE AbteilungsNr = 1
```

```
SELECT AbteilungsNr, COUNT(abtnr) FROM Personal WHERE AbteilungsNr = 2
usw.
```

Mit jeder Anweisung erhalten Sie die Anzahl der Mitarbeiter in der jeweiligen Abteilung. Diese Vorgehen ist recht umständlich und bei mehreren Abteilungen auch ziemlich aufwendig. Das gleiche Problem läßt sich mit der GROUP BY-Anweisung sehr elegant lösen:

```
SELECT AbteilungsNr, COUNT(AbteilungsNr)
FROM Personal
GROUP BY AbteilungsNr
```

Die Bearbeitung des obigen Befehls können Sie sich wie folgt vorstellen:

Auswahl aller Datensätze der Tabelle Personal (FROM Personal),
Gruppieren der Datensätze nach Abteilungsnummer (GROUP BY AbteilungsNr),
Zählen der Datensätze innerhalb der gebildeten Gruppen (... COUNT(AbteilungsNr)).

c) Um die Nettosummen aller Rechnungen zu bestimmen, brauchen Sie die Tabellen Rechnungsdaten und Artikel. Der Artikeltabelle wird der Preis entnommen, die Tabelle Rechnungsdaten enthält alle Artikel, die zu einer Rechnung gehören:

```
SELECT r.RechnungsNr, SUM(Artikel.Verkaufspreis*r.Artikelanzahl)
FROM Rechnungsdaten r,  Artikel
WHERE r.ArtikelNr = Artikel.Nr.
GROUP BY RechnungsNr
```

Dabei passiert folgendes:

Auswahl aller Einträge der Tabelle Rechnungsdaten
RIGHT JOIN der Tabelle Artikel, wobei die Artikelnummer übereinstimmen müssen
Gruppieren der Datensätze nach Rechnungsnummer
Die Produkte aus Verkaufspreis und Artikelanzahl werden für jede Rechnungsnummer addiert (Bilden der Nettosumme)
Ausgabe von Rechnungsnummer und Nettosumme

6. Begrenzung der Ausgabe mit HAVING

Häufig benötigt man nicht alle Datensätze, die das Ergebnis einer SELECT-Anweisung sind, sondern nur solche, die bestimmten Bedingungen erfüllen. Das lässt sich mit HAVING realisieren. Wenn z.B. nur die Rechnungsnummern mit Nettosumme angezeigt werden sollen, bei denen die Rechnungssumme 1.000 Euro überschreitet, kann man die o.a. Anweisung wie folgt erweitern:

```
SELECT r.RechnungsNr, SUM(Artikel.Verkaufspreis*r.Artikelanzahl)
FROM Rechnungsdaten r,  Artikel
WHERE r.ArtikelNr = Artikel.Nr.
GROUP BY RechnungsNr
HAVING SUM(Artikel.Verkaufspreis*r.Artikelanzahl) >1000
```

7. Sortieren von Tabellen mit ORDER BY

In den bisherigen Abfragen wurden die Daten ausschließlich in ungeordneter Folge ausgegeben. Mit einer ORDER BY-Klausel können Sie den Sortierbegriff (z. B. Name) sowie die Sortierfolge (auf-/absteigend) festlegen. Syntax:

```
SELECT ...
FROM ...
[ORDER BY SpaltenName [ASC|DESC] {, SpaltenName [ASC|DESC]}]
```

Der Parameter ASC steht dabei für aufsteigend (z.B. die übliche Sortierung nach dem Alphabet), DESC für absteigend.
Die Angabe mehrerer Sortierbegriffe ist z. B. dann sinnvoll, wenn Sie Personennamen sortieren und als zweiten Sortierbegriff den Vornamen angeben. Beispiel:

```
SELECT *
FROM Artikel
ORDER BY Nr
```

... sortiert die Artikel in aufsteigender Reihenfolge.

Hinweis: Entscheidend für die Ausführungsgeschwindigkeit ist die Indizierung der entsprechenden Felder. Das Sortieren einer größeren Datenbank ohne Index zwingt Sie zu einer ungewollten Kaffeepause.

8. Daten manipulieren mit UPDATE

Eine Möglichkeit der Datenmanipulation, z. B. die Erhöhung des Artikelpreises, haben Sie bereits kennengelernt. Sie mußten „manuell“ alle Datensätze durchlaufen und das entsprechende Feld ändern. Eine bedeutend einfachere Möglichkeit bietet sich mit dem UPDATE-Befehl. In Kombination mit einer WHERE-Klausel lassen sich schnell alle gesuchten Datensätze ändern. Diese Änderung kann sich auf mehrere Felder eines Datensatzes auswirken.
Syntax:

```
UPDATE <Tabellename> SET <Feld> = <Ausdruck> WHERE ...
```

Für den mathematischen Änderungsausdruck steht Ihnen auch der bisherige Feldinhalt zur Verfügung.
Erste Möglichkeit: Preis = Preis * 1.10 für eine 10 % Preiserhöhung
Zweite Möglichkeit: Preis = einkaufspreis * 1.3

a) Die Realisierung:

```
UPDATE artikel
SET verkaufspreis = einkaufspreis * 1.1
```

... berechnet den Verkaufspreis für alle Artikel aus dem Einkaufspreis mit 10% Aufschlag.

b) Möchten Sie nur einige Felder verändern, setzen Sie die WHERE-Klausel ein:

```
UPDATE artikel
SET verkaufspreis = einkaufspreis * 1.1
WHERE verkaufspreis > 200
```

c) Um mehrere Felder gleichzeitig zu ändern, trennen Sie die entsprechenden Ausdrücke mit einem Komma:

```
UPDATE artikel
SET verkaufspreis = verkaufspreis * 1.1, einkaufspreis = einkaufspreis * 1.1
WHERE verkaufspreis > 200
```

9. Löschen von Datensätzen mit DELETE

Kurz und bündig: Datensätze löschen Sie mit der DELETE-Anweisung. Syntax:

```
DELETE FROM <Tabellenname> WHERE <logischer Ausdruck>
```

Der DELETE-Befehl läßt sich natürlich mit entsprechenden WHERE-Klauseln kombinieren.