SQL - Entstehung und Entwicklung

SQL wurde im Rahmen eines IBM-Forschungsauftrags Mitte der siebziger Jahre entwickelt. Sei der Markteinführung von SQL im Jahr 1979 haben zahlreichen Unternehmen SQL als Standard-Datenbanksprache sowohl für Großrechner- als auch für Minicomputerumgebungen übernommen. Die Einführung von SQL bei Mikrocomputern bedeutet, daß Unternehmen das Entwickeln von Datenbankanwendungen weitgehend standardisieren können.
SQL ist die einzige standardisierte Datenbanksprache auf der Basis des relationalen Datenmodells (ANSI X3.135/1986 - ISO 907/1987 - DIN 66315). Man kann sie als Datenbank-Subsprache Mit Möglichkeiten zur Definition, Manipulation und Integritätskontrolle von Datenbeständen bezeichnen. Hierbei wird von einer Mehrbenutzer-Datenbank als Normalfall ausgegangen, so daß abgestufte Zugriffsberechtigungen ebenso selbstverständlich sind wie die Möglichkeiten, spezifische Benutzersichten (*views)* auf einer oder mehrerer Tabellen zu definieren.
SQL ist eine fortgeschrittene Sprache für relationale Datenbanken, in denen Daten generell als logische Mengen, genannt Relationen, behandelt werden. In einer relationalen Datenbank werden mit einer entsprechenden Datenbanksprache sämtliche Daten in einer einzigen bzw. in einer Menge von Tabellen definiert. SQL besteht aus einem kleinen Satz präziser Befehle, die es ermöglichen, Informationen in Tabellen zu definieren, anzuzeigen und zu aktualisieren. Durch Reduzierung der Anzahl von Befehlen für den Datenzugriff bewirkt SQL eine Verkürzung des Zeit- und Programmieraufwands zur Durchführung komplexer Abfragen. SQL kann auch den beim Ändern einer Datenbankanwendung anfallenden Aufwand verringern.

SQL-Sichten

Eine SQL-Sicht ist eine Untermenge von Zeilen und Spalten von einer oder mehreren vorhandenen Tabellen. Sichten werden häufig auch als virtuelle Tabellen bezeichnet, da sie in Wirklichkeit keine Daten enthalten. Die Sicht ist vielmehr die Abbildung der in einer oder mehreren zugrundeliegenden Ausgangstabellen enthaltenen Daten. Die in einer Sicht angezeigten Daten sind dynamisch. Dies bedeutet, daß bei Änderung der in einer Ausgangstabelle enthaltenen Daten auch die Daten in der entsprechenden Sicht aktualisiert werden. Das gleiche gilt für den umgekehrten Vorgang: Wenn die Daten einer Sicht geändert werden, werden die Daten in den Ausgangstabellen ebenfalls aktualisiert.

SQL - Die Sprache

SQL unterscheidet sich von den prozeduralen Programmiersprachen (einschließlich der Programmiersprachen aus dem Umfeld von dBASE, die herkömmlich satzorientiert arbeiten) dadurch, daß keine Variablen und auch keine der üblichen Kontrollstrukturen wie IF ... THEN ... ELSE oder WHILE ... DO zur Verfügung stehen. Charakteristisch für SQL ist die mengenorientierte Arbeitsweise: z.B. vermittelt ein mit SELECT konstruierter Befehl eine Abbildung von einer oder mehrerer (in der Regel über joins verbundener) Relationen und liefert als Ergebnis wieder eine Relation. Da man - im Gegensatz zu prozeduralen Sprachen - nicht angeben muß, wie eine bestimmte Information zusammengestellt werden soll, sondern nur formuliert werden muß, was man wissen will, wird SQL zu den deklarativen Sprachen gerechnet. Manchmal wird sie deshalb auch als Programmiersprache der 4. Generation bezeichnet, obwohl mit 4GL sonst eher SQL-Erweiterungen gemeint sind. Bei der schnellen Entwicklung in der Datenverarbeitung muß SQL als „Oldie“ gelten, denn diese Sprache wurde bereits Anfang der 70er Jahre entwickelt. Man stellte sich damals vor, daß jeder Sachbearbeiter mit einem SQL-Interpreter umgehen könne. Heute wird SQL eher als Sprache für Anwendungsentwickler verwendet, da inzwischen die Ansprüche an den Komfort der Benutzerschnittstellen deutlich gestiegen sind.
SQL besteht aus weniger als 30 Befehlen. Diese ermöglichen jedoch die Durchführung sämtlicher notwendigen Operationen, um Tabellen und Sichten zu definieren und darin enthaltene Daten abzufragen, zu aktualisieren, zu löschen oder neue Daten einzufügen. SQL-Befehle sind einfach zu verwenden, da sich alle auf die gleichen Elemente beziehen - die Zeilen und Spalten einer Tabelle oder einer Sicht.

Beispiel für eine SQL-Abfrage

```
SELECT Art_Nr, Art_Bez, Lagerbest, Kosten
FROM Inventar
WHERE Lagerbest > 3;
```

In diesem Beispiel zeigt das Ergebnis dieser Abfrage in einer vierspaltigen alle Artikel (einschließlich deren Bezeichnung) aus der Tabelle Inventar. SELECT legt fest, welche Spalten angezeigt werden sollen. Die Klausel FROM bestimmt die Tabelle oder Tabellen, aus denen die Daten angezeigt werden sollen. Die WHERE Bestimmung legt die Bedingung fest: Es werden nur die Artikel angezeigt, von denen mehr als 3 Stück vorhanden sind.
SQL-Befehle können in dBASE direkt (interaktiv) im Befehlsfenster eingegeben werden. Sie können aber auch in dBASE-Programmdateien eingebunden werden. Auch in anderen Sprachen, wie z.B. Fortran, Cobol oder C steht in der Regel eine Schnittstelle zur Verfügung, mit der man SQL-Befehle einbinden kann.
dBASE bietet zwei Modi für die Befehlseingabe. Der Standard-Modus ermöglicht nur die Eingabe der gebräuchlichen dBASE-Befehle; der zweite Modus ermöglicht die Verwendung der SQL-Befehle, schränkt jedoch die Verwendung von dBASE-Befehlen ein. Das Umschalten in den SQL-Modus ist notwendig, weil SQL und dBASE unterschiedliche Wirkungswesen haben, bestimmte dBASE- und SQL-Befehle aber identisch sind.

SQL im interaktiven Modus

Die einfachste und schnellste Methode, um die Arbeit mit SQL zu beginnen, ist das Ausführen von Befehlen im interaktiven Modus. Bei der interaktiven Ausführung von SQL-Befehlen erhält man sofort eine Rückmeldung vom System: Nach Eingabe eines SQL-Befehls erscheint sofort das Ergebnis.
Um den SQL-Modus von dBASE zu aktivieren, wird im Befehlsfenster der Befehl SET SQL ON eingegeben. In der Statuszeile wird SQL angezeigt. Zum Verlassen des SQL-Modus wird SET SQL OFF eingegeben.
Alle SQL-Befehle werden im interaktiven Modus mit einem Semikolon abgeschlossen. Ein SQL-Befehl kann über mehrere Zeilen hinausgehen, darf aber nicht mehr als 1024 Zeichen haben. (vgl. unten: SQL-Befehle eingeben). Das Befehlsfenster verfügt über einen Befehlspuffer, der automatisch die eingegebenen Befehle speichert. Um einen bereits vorher eingegebenen Befehl noch einmal zu starten, genügt es, den Cursor in die betreffende Zeile zu setzen und RETURN zu drücken.

SQL - Datenbanken anlegen

Eine SQL-Datenbank muß zunächst definiert werden. Ein dBASE-Katalog läßt sich in diesem Fall nicht übernehmen. Das Anlegen einer SQL-Datenbank bedeutet das Anlegen eines Unterverzeichnisses mit dem Namen dieser Datenbank. Dies geschieht mit dem Befehl:

CREATE DATABASE [<Suchpfad>] <Name der Datenbank>;

Vorhandene Datenbanken lassen sich anzeigen mit dem Befehl SHOW DATABASE. In dem damit geschaffenen Unterverzeichnis werden eine Reihe von Katalogtabellen angelegt, in denen sämtliche Datenbanktabellen aufgezeichnet werden, die erstellt wurden, während die Datenbank geöffnet war. Wenn mit dBASE oder einem anderen Programm erstellte Bestandsdateien überommen werden sollen, müssen diese in das neu erstellte (oder vorhandene) Unterverzeichnis kopiert werden. Dann wird im Befehlsfenster die Datenbank gestartet mit dem Befehl:

START DATABASE [<Name der Datenbank>]

Mit dem Befehl START DATABASE werden alle Tabellen in der Datenbank in einen geöffneten dBASE-Katalog (.CAT) eingetragen, sofern der Befehl SET CATALOG nicht auf OFF gesetzt ist. SQL-Anweisungen lassen sich erst eingeben, wenn eine Datenbank geöffnet ist. In SQL kann immer nur eine Datenbank geöffnet sein. Wird eine weitere geöffnet, so wird automatisch die bisher bearbeitete Datenbank geschlossen. Um nicht mit SQL erstellte Bestandsdateien in die Datenbank aufzunehmen muß jetzt jede Datei aufgerufen werden mit dem Befehl:

DBDEFINE <Name der Bestandsdatei>

Durch diesen Aufruf wird der Name der Bestandsdatei sowie deren Struktur in die entsprechenden Katalog-dateien eingetragen. Dabei sind allerdings eine Reihe von Einschränkungen zu beachten. (s. Abschnitt „Externe Daten übertragen“

SQL-Befehle eingeben

Die bisherigen SQL-Befehle waren sehr kurz und lassen sich am schnellsten direkt in das Befehlsfenster eingeben. Die folgenden SQL-Befehle benötigen in der Regel mehrere Zeilen. Da das Befehlsfenster aber nur einzeilige Befehle (bis zu 255 Zeichen) verarbeitet, müssen die folgenden Befehle mit dem Programmeditor geschrieben werden. Dazu wird aus dem Menü *Datei* die Option *Datei öffnen* ausgewählt. Ein beliebig wählbarer Dateiname muß dabei die Endung .PRS haben, damit dBASE bei der Verarbeitung der Programm-datei automatisch in den SQL-Modus umschaltet. Nach Eingabe der SQL-Befehle wird mit STRG-F9 die Programmverarbeitung getartet. In der Regel ist ein Absichern der Programmdatei nicht notwendig.

Create Table

Über den Befehl CREATE TABLE wird in der aktuellen Datenbank eine neue SQL-Tabelle angelegt. Eine besondere Berechtigung ist dazu nicht erforderlich. Ist der Zugriff auf dBASE kennwortgeschützt, werden neue Tabellen nach dem Anlegen verschlüsselt. Die Syntax lautet:

```
CREATE TABLE <Tabellenname>
    (<Spaltenname> <Datentyp> [,<Spaltenname> <Datentyp>...]);
```

SQL unterscheidet folgende Datentypen

- SMALLINT — Eine Maximal 6-stellige Zahl (einschließlich Vorzeichen). Zulässige Eingabewerte - 99999 bis 999999. Dieser Spaltentyp wird in der Bestandsdatei zu einem numerischen Feld NUMERIC(6,0)
- INTEGER — 11-stellige Zahl, ansonsten wie oben
- DECIMAL(x,y) — x-stellige Dezimalzahl (einschließlich Vorzeichen) im Festkommaformat und y Dezimalstellen. Für x können Werte von 1 bis 19, für y Werte von 0 - 18 eingesetzt werden, entspricht NUMERIC(x+1,y)
- NUMERIC(x,y) — Eine x-stellige Dezimalzahl (einschließlich Vorzeichen und Dezimalkomma) im Festkommaformat und y Dezimalstellen. Für x können Werte von 1 bis 20, für y Werte von 0 bis 18 gewählt werden, entspricht NUMERIC(x,y).
- FLOAT(x,y) — entspricht dem dBASE-Typ FLOAT(x,y).
- CHAR(n) — entspricht dem dBASE-Typ ZEICHEN(n).
- DATE — entspricht dem dBASE-Typ Datum
- LOGICAL — entspricht dem logischen Typ in dBASE. Die Werte werden durch die Konstanten .T., .t.,.Y.,.y.,.F.,.f.,.N.,.n. angegeben.

In einer Tabelle können nicht mehr als 255 Spalten angelegt werden; die Gesamtbreite einer Tabelle darf 4.000 Byte nicht überschreiten. In SQL von dBASE für DOS können keine Memofelder angelegt oder bearbeitet werden. Ein Spaltenname kann maximal 10 Zeichen lang sein. Ein Beispiel für CREATE TABLE: Es soll eine Tabelle mit dem Namen „Versand“ angelegt werden, in der alle von der Firma versendeten Waren aufgeführt sind:

```
CREATE TABLE Versand
 (Vers_nr      CHAR(6),
  Vers_Dat     DATE,
  Auftr_Nr     CHAR(6),
  Sped         CHAR(25),
  Gew          DECIMAL(4,1));
```

INSERT INTO

Mit dem Befehl INSERT können in eine Tabelle oder in eine aktualisierbare Sicht Zeilen (d.h. Datensätze eingefügt werden. Wenn der Zugriff auf dBASE kennwortgeschützt ist, wird für das Hinzufügen von Datensätzen die Zugriffsberechtigung INSERT benötigt. Die Syntax von INSERT zeigt zwei Varianten:

1:
```
INSERT INTO <Tabellenname>
    [(<Spaltenliste>)]
    VALUES (<Werteliste>);
```

2:
```
INSERT INTO <Tabellenname>
    [(<Spaltenliste>)]
    <SELECT-Befehl>;
```

Für die Werte einer Werteliste gelten folgende Regeln:

- Es können Konstanten, Speichervariablen, oder Funktionen, die einen Wert liefern, sein.
- Sie müssen durch Kommas getrennt werden. Zeichenfolgenmüssen zwischen halbe oder doppelte Anführungszeichen, Datumswerte zwischen geschweifte Klammern gesetzt werden.
- Sie müssen denselben Datentyp haben, wie die Spalte, zu der sie hinzugefügt werden.

Wenn nicht in jede Spalte der Tabelle der Tabelle oder Sicht ein Wert eingefügt werden soll, werden die nicht angegebenen Werte folgendermaßen initialisiert:

- Eine alphanumerische Spalte erhält ein Leerzeichen.
- Eine numerische Spalte erhält den Wert Null.
- Eine Datumsspalte erhält eine leere Datumsfolge (im Format TT..MM..JJ).
- Eine logische Spalte erhält den Wert .F.

In der <Spaltenliste> können die Spaltennamen der Tabelle oder Sicht angegeben werden, in die Zeilen eingefügt werden sollen. Die Spaltenliste kann aber auch weggelassen werden, wenn die Reihenfolge der Werte in der Werteliste oder die Reihenfolge der Spalten im Befehl SELECT dieselbe ist wie in der Tabelle oder Sicht, in die Zeilen eingefügt werden sollen. Eine Spaltenliste muß allerdings angegeben werden, wenn nicht für jede Spalte einer Tabelle oder Sicht ein Wert festgelegt werden soll. Eine Spaltenliste muß nicht der Reihenfolge der Spalten in der Tabelle entsprechen. Die Spaltenliste muß aber dieselbe Reihenfolge haben wie die Werteliste.

Beispiele:

Es existiert eine Tabelle Personal mit den Spalten: Pers_nr, Nachname, Vorname, Einst_dat, Einsatz, vorgesetzt, Gehalt, Provision. Dann könnte die Eingabe eines Datensatzees folgendermaßen aussehen.

1.
```
INSERT INTO Personal
VALUES (“0001“,“Winter“,“Helmut“,{15.06.88},
“München“,“00008“,4200,3.0);
```

In die gleiche Tabelle lassen sich aber auch Einzelwerte eintragen:

2.
```
INSERT INTO Personal
(Gehalt, Provision)
VALUES (5800,8.5);
```

Es lassen sich mit INSERT aber auch Datensätze oder einzelne Spalten aus einer vorhandenen Datei eintragen:

3.
```
INSERT INTO Person1
        SELECT *
        FROM Personal
        WHERE Nachname = “Winter“
```

Mit UPDATE Daten aktualisieren

Der SQL-Befehl UPDATE ermöglicht es, die Werte in einer oder mehreren ausgewählten Zeile zu aktualisieren. Die Befehlssyntax lautet:

```
UPDATE <Tabellenname>
    SET <Spaltenname> = <Ausdruck>
    [,<Spaltenname> = <Ausdruck> ...]
    [ WHERE - <Klausel> ];
```

Angenommen, es existiert eine Tabelle *schueler* mit den Schülerdaten von 24 Schülern einer 7.Klasse. Am Schuljahresende sollen die Spalte „Schuljahr“ aktualisert werden und in 8G umgewandelt werden, falls der betreffende Schüler in der Spalte versetzt mit einem “j“ vermerkt ist. In diesem Fall würde man eingeben:

```
UPDATE schueler
    SET schuljahr = “8G“
    WHERE versetzt = “j“;
```

Es erscheint die Meldung: *24 Zeilen aktualisiert.*
In diesem Beispiel legt die WHERE-Klausel eine einfache Bedingung fest. Komplexere Bedingungsselektionen lassen sich mit dem SELECT-Befehl festlegen. (s. Abschnitt SELECT)

Mit DELETE Daten löschen

Mit dem Befehl DELETE können ausgewählte Zeilen gelöscht werden. Die Befehlssyntax lautet:

```
DELETE FROM <Tabellenname>
    [WHERE <Klausel>];
```

Soll zum Beispiel aus der oben erwähnten Tabelle ein Schüler gelöscht werden, so wird folgende Anweisung eingegeben:

```
DELETE FROM schueler
    WHERE name = “Musterschüler“;
```

Es erscheint die Meldung 1 Zeile(n) gelöscht.

!! Wird bei der Eingabe des DELETE-Befehls die WHERE-Klausel nicht festgelegt, so werden alle Datensätze dieser Tabelle gelöscht.

Mit ALTER die Tabelle ändern

Sollte es sich als notwendig erweisen, die Struktur einer Tabelle zu ändern, so ist das bei SQL-Tabellen nicht möglich mit der Option „Ändern“ im Regiezentrum. Stattdessen wird der Befehl ALTER verwendet, mit dem sich eine oder mehrere Spalten hinzufügen lassen. Die Syntax folgendermaßen lautet:

```
ALTER TABLE <Tabellenname>
    ADD <Spaltenname> <Datentyp>
    [, <Spaltenname> <Datentyp>...];
```

```
Beispiel:  ALTER TABLE schueler
             ADD (Telefon CHAR(13));
```

Es erscheint die Meldung: *TELEFON hinzugefügt zu Tabelle: SCHUELER*. Die neue Spalte wird hinter der letzten vorhandenen Spalte angeordnet.

Der SELECT-Befehl

Der Select-Befehl ist der komplexeste Befehl von SQL. Er wird sowohl im interaktiven als auch im eingebundenen Modus für die Datenabfrage verwandt. Mit einer einzigen SQL-Anweisung läßt sich eine beliebige Zusammenstellung von Daten aus einer oder mehreren Tabellen anzeigen. Zur Befehlssyntax gehören auch die im folgenden aufgelisteten Klauseln:

```
SELECT <Klausel>
    [INTO <Klausel>]
    FROM <Klausel>
    [WHERE <Klausel>]
    [GROUP BY <Klausel>]
    [HAVING <Klausel>]
    [UNION Unterauswahl] ...
    [ORDER BY <Klausel> /FOR UPDATE OF <Klausel>]
    [SAVE TO TEMP <Klausel>]
```

Zum Erstellen von SQL-Abfragen lassen sich die Klauseln in dieser Reihenfolge miteinander kombinieren.Die Klauseln INTO und FOR UPDATE OF werden im eingebundenen SQL-Modus zu Gestalten von Abfragen verwendet. Eine Unterauswahl ist eine weitere SELECT-Auswahl, die durch die UNION-Klausel „aneinandergereiht“ wird.

Die einfachste Form des Befehls SELECT, mit der bestimmte Spalten aus einer einzelnen Tabelle ausgewählt und angezeigt werden können, ist folgende:

```
SELECT <Spalten>
    FROM <Tabellen>;
```

Will man aus der Tabelle Schueler alle Vornamen und Nachnamen haben so lautet die SELECT-Anweisung:

```
SELECT Vorname, Nachname
    FROM schueler;
```

Das Ergebnis einer SELECT-Abfage wird *Ergebnistabelle* genannt. Eine Ergebnistabelle wird nicht abgespeichert, es sei denn, der SELECT-Befehl enthält die Anweisung SAVE TO TEMP <Klausel>. Sollen alle Spalten angezeigt werden, so läßt sich auch der Joker * verwenden:

```
SELECT * FROM schueler;
```

Bedingung WHERE in SELECT-Anweisungen

Die Tabelle *belegt* beinhaltet die Kursnummern, die Schülernummern, die jeweils einen Kurs belegt haben und die Lehrernummern, die diesen Kurs halten. Will man alle Kurse, die der Schüler mit der Schülernummer (S_nr) = 001 hat auflisten, lautet der SELECT-Befehl:

```
SELECT K_nr
    FROM belegt
    WHERE S_nr = 001;
```

```
Syntax: SELECT <Spaltenliste>
            FROM <Tabellen>
            WHERE <Bedingung>
```

Als Operatoren kommen in SQL folgende Symbole in Betracht:

- = Gleichheit
- < Kleiner als
- > Größer als
- <= Kleiner oder gleich
- >= Größer oder gleich
- <> Ungleich
- ! Negation der Vergleichsoperation < oder > oder =

Miteinander zu vergleichende Werte müssen demselben Datentyp angehören. Dabei sind SMALLINT, INTEGER, DECIMAL und FLOAT allerdings miteinander vergleichbar.

Weitere Operatoren sind:

- der Intervall-Operator: WHERE eingang BETWEEN 1988 AND 1990
- Test auf NULL WHERE telefon IS NULL
- IN-Operator WHERE bezirk IN (“Reinickendorf“ , “Wedding“)
- Ähnlichkeitsoperator WHERE preis LIKE ‘8%’ { Wildcard }
- EXIST-Operator WHERE EXISTS (SELECT * FROM zweittabelle WHERE …)

GROUP-BY-Klausel

In manchen Fällen - insbesondere bei numerischen Auswertungen - sind zwei Konstruktionen nützlich, die die Syntax der SELECT-Anweisung ‘anreichern’: die GROUP BY - Klausel und mit ihr die HAVING - Klausel. Mit der GROUP BY-Klausel werden in der Ergebnistabelle jeweils alle Zeilen zusammengefaßt, die in einer bestimmten Spalte den gleichen Wert aufweisen. Jede Gruppe wird zu einer Zeile zusammengefaßt. Die Gruppierungsklausel wird meist bei der Verwendung statistischer Funktionen auf Spalten nötig. Jeder der hinter GROUP BY angegebenen Spaltenbezeichner muß vor der FROM-Anweisung ebenfalls aufgeführt sein.

Beispiel: SELECT s_nr, MAX (punkte) Aus der Tabelle beleg soll für jeden Schüler die
FROM beleg beste Punktzahl ermittelt werden, die er in einem
GROUP BY s_nr Kurs erhalten hat.

Das Ergebnis ist nicht notwendigerweise sortiert.

Having - Klausel

Im Zusammenhang mit der Gruppierungsklausel tritt häufig die HAVING-Klausel auf. Durch eine logische Bedingung nach dem Schlüsselwort HAVING werden bestimmte Gruppen selektiert. Die HAVING-Klausel ist vergleichbar mit der WHERE-Klausel bei SELECT

BEISPIEL SELECT s_nr, MIN (punkte) Es wird für jeden Schüler die schlechteste
FROM beleg Punktzahl ermittelt sofern er in einem Kurs
GROUP BY s_nr weniger als 5 Punkte erzielt hat.
HAVING MIN (punkte) < 5

Mit HAVING wird einschränkend bestimmt, welche Zeilen in einem Gruppenergebnis angezeigt werden. Folgende Regeln gelten für die Anwendung von HAVING:

- Jede Suchbedingung der Klausel HAVING entspricht im allgemeinen einer Formelfunktion, die in der SELECT-Klausel angegeben ist.
- Die Klausel FROM einer SELECT-Unterabfrage, die in einer Suchbedingung der Klausel HAVING verwendet wird, darf nicht auf eine Tabelle oder Sicht verweisen, die in der FROM-Klausel der SELECT-Hauptabfrage angegeben ist.
- Eine korrelierende Unterabfrage darf nicht in einer Suchbedingung der HAVING-Klausel verwendet werden.

Mit ORDER BY die Ergebnisausgabe sortieren

Mit einer WHERE-Klausel werden im Ergebnis nur jene Zeilen angezeigt, die der Bedingung der WHERE-Klausel entsprechen. Durch Hinzufügen einer ORDER BY-Klausel lassen sich die Zeilen selektieren, die angezeigt werden sollen, wobei diese Zeilen mit Hilfe einer einzigen SELECT-Anweisung in einer bestimmten Reihenfolge geordnet werden können. Beispiel:

SELECT S_nr, K_nr, klausur Ergebnis: Die Schülernummern, die Kursnummern und
FROM belegung Klausurergebnis werden ausgegeben, sortiert
WHERE K_nr = ‘GIN-3’ nach den Klausurergebnissen
ORDER BY klausur

Mit CREATE VIEW Sichten erstellen

Zum Erstellen von Sichten wird der Befehl CREATE VIEW verwendet. Die Befehlssysntax lautet:

```
CREATE VIEW <Sichtname> [(<Liste der Spaltennamen>)]
  AS <SELECT-Anweisung>
  [WITH  CHECK OPTION];
```

Der Befehl CREATE VIEW besteht aus drei Teilen. Im ersten Teil wird der Name der Sicht angegeben. Wahlweise können hier auch Spaltennamen (für die Sicht) angegeben werden. Wenn keine Spaltennamen angegeben werden, erscheint die Liste der Splatennamen für die Sicht in der SELECT-Anweisung. Wenn jedoch die Werte in bestimmten der Sicht von dBASE-Funktionen oder anderen Ausdrücken abhängen, muß eine Liste mit Spaltennamen für die Sicht eingegeben werden.
Der zweite Teil von CREATE VIEW enthält die SELECT-Anweisung, die festlegt, welche Zeilen und Spalten aus der (den) Ausgangstabelle(n) in die Sicht aufgenommen werden.
Der dritte Teil von CREATE VIEW, die WITH CHECK OPTION, ist wahlfrei. Sie wird bei Sichten verwendet, die aktualisiert werden können, und legt fest, daß eingefügte oder aktualisierte Zeilen auf ihre Übereinstimmung mit der Sichtdefinition geprüft werden.
Durch Verwendung der Option wird sichergestellt, daß nur solche Zeilen in die Sicht eingefügt und dain angezeigt werden, die einer in der WHERE-Klausel (in der SELCT-Anweisung) festgelegten Bedingung entsprechen. Beispiel:

```
CREATE VIEW muenchen
    AS SELECT PersNr, Nachname, Einst_Dat
    FROM person1
    WHERE einsatz = 'Müchen';
```

Aus der Tabelle person1 wird eine Sicht erstellt, die nur die Zeilen der München angestellten Mitarbeiter und nur die drei Spalten PersNr, Nachname, Einst_Dat enthält.

Es erscheint eine Meldung *Sicht MUENCHEN erstellt.* Nachdem diese Sicht erstellt wurde, können die Zeilen dieser Sicht mit einer einfachen SELECT-Anweisung abgefragt werden:

```
SELECT *
    FROM muenchen
```

Ergebnis:

PERSNR	NACHNAME	EINST_DAT
0001	Winter	15.06.89
0003	Scholl	12.03.89
0007	Hartmann	01.04.94

Mit Sichten eine Datenbank umstrukturieren

Mit Hilfe von Sichten können Informationen in einer Datenbank neu strukturiert werden, ohne dazu neue Tabellen mit geänderten Spaltendefintionen anlegen zu müssen. Wenn z.B. eine Sicht erstellt werden soll, in der die Spalten Nachname und Vorname der Tabelle SCHUELER zu einem für den Ausdruck geeigneteren Format zusammengeführt werden sollen, so läßt sich das mit folgender Sicht bewerkstelligen:

```
CREATE VIEW adresse
    (vollname, ort, plz)
    AS SELECT vorname+nachname, ort, plz
    FROM schueler;
```

In diesem Beispiel werden die Spalten VORNAME und NACHNAME zusammengeführt und in der SICHT in der Spalte VOLLNAME angezeigt.

Mit DROP VIEW Sichten löschen

Sichten werden gelöscht,

- wenn man eine Ausgangstabelle für die Sicht löscht,
- die Datenbank löscht,
- den Befehl DROP VIEW <Sichtname> verwendet.