In diesem Abschnitt werden drei Anweisungen vorgestellt, mit denen Daten manipuliert werden können:

- ➔ INSERT Daten einfügen
- ➔ UPDATE Daten aktualisieren
- ➔ DELETE Daten löschen

1. Dateneingabe

Um eine Relation mit Tupeln zu füllen, stellt SQL die Anweisung INSERT zur Verfügung.

INSERT-Anweisung

Die Grundform dieser Anweisung lautet:

```
INSERT INTO relation (attributliste)
   VALUES (werteliste);
```

SQL erlaubt es darüber hinaus, mehrere Datensätze mit einer INSERT-Anweisung zu erfassen. Die Klammer hinter der VALUES-Klausel muss dann für jedes Tupel wiederholt werden. Die formale Syntax lautet für diesen Fall:

```
INSERT INTO relation (attributliste)
   VALUES (werteliste) {, (werteliste)};
```

Die Anzahl der Angaben in der Werteliste muss mit der Anzahl der Angaben in der Attributliste übereinstimmen. Die Zuordnung der Werte zu den Attributen erfolgt über die Reihenfolge, d.h. der an dritter Stelle angegebene Wert wird in das dritte angegebene Attribut der Relation eingetragen.

Grundsätzlich muss für jedes Attribut, für das keine Nullmarke zulässig ist und das keine DEFAULT -Klausel enthält, ein Wert bereitgestellt werden.

In die Relation T_Kunden können wir beispielsweise einen Datensatz durch folgende Anweisung einfügen:

```
INSERT INTO T_Kunden ( Kunden_Nr, Status, Vorname, Nachname, Strasse, PLZ, Ort,
                       letzteBestellung, letzteWerbeaktion, Zahlungsart )
              VALUES (199, 'S', 'Klaus', 'Meier', 'Weserstr. 12', '10999', 'Berlin',
                      '1996-04-28', '1995-04-13', 'N');
```

Die Attribute, die bei der Aufzählung der Spalten nicht genannt werden, erhalten automatisch den Default-Wert, der bei der Definition der Relation festgelegt wurde. Implizit ist dies in vielen Fällen NULL. Soll in einer Spalte explizit eine Nullmarke eingetragen werden, so wird in der VALUES-Klausel das Schlüsselwort NULL dafür eingesetzt, wie in folgendem Beispiel, wo das Lieferdatum noch nicht bekannt ist:

```
INSERT INTO T_Bestellungen(Bestell_Nr, F_Kunden_Nr, Bestelldatum, Lieferdatum)
        VALUES (960200, 100, '2004-02-24', NULL);
```

Die in der VALUES-Klausel enthaltenen Werte müssen alle für die Relation definierten Integritätsbedingungen einhalten, sonst wird die Operation vom DBMS abgebrochen. Der effektivere und benutzerfreundlichere Weg, Daten einzugeben oder zu korrigieren, ist sicherlich die Benutzung einer komfortablen Eingabemaske. Es gehört jedoch nicht zu den Aufgaben von SQL, die entsprechenden Hilfsmittel zur Verfügung zu stellen.

Zur Programmierung oder Generierung von Anwendungsprogrammen gibt es unzählige Möglichkeiten. Die ganze Welt der Client/Server-Entwicklungswerkzeuge steht hier zur Verfügung. Datenbankhersteller bieten teilweise selbst Programmiersprachen oder

Programmgeneratoren an, die wiederum teilweise nur mit den eigenen, teilweise auch mit fremden Datenbankmaschinen zusammenarbeiten können.

Irgendwo werden aber alle diese Werkzeuge die vom Benutzer im Bildschirmdialog erfassten Daten mit der Anweisung INSERT INTO in die Datenbank übertragen, auch wenn der Anwender davon nichts sieht.

Eine Variante der INSERT-Anweisung erlaubt das Übertragen von Daten aus einer oder mehreren Tabellen, die als Abfrage formuliert sind. Sie folgt der Syntax:

```
INSERT INTO relation (attributliste)
    abfrage;
```

Diese Form wird beispielsweise benötigt, um Kopien (Snapshots) von Relationen zu erzeugen.

Effekte der Integritätsbedingungen

Wegen der Eindeutigkeit des Primärschlüssels wird die nochmalige Eingabe eines bereits definierten Primärschlüsselwerts durch das DBMS verhindert.

Beispiel:

```
INSERT INTO T_Kunden ( Kunden_Nr, Status, Vorname, Nachname, Strasse, PLZ, Ort,
                       letzteBestellung, letzteWerbeaktion, Zahlung )
    VALUES (102, 'S', 'Hans', 'Moser', 'Wiesenstr. 8', '12444', 'Zerlitz', '2001-01-01', '2001-01-20',
    'N');
```

Das Datenbanksystem erzeugt dann eine Fehlermeldung, die zum Inhalt hat, dass die Eindeutigkeit des Primärschlüssels verletzt ist.

Der Versuch, eine Bestellung mit ungültiger Kundennummer zu erfassen, wie in

```
INSERT INTO T_Bestellung (Bestell_Nr, F_Kunden_Nr, Bestelldatum, Lieferdatum)
    VALUES (960300, 577, '2004-03-30', NULL);
```

wird ebenfalls zurückgewiesen und vom DBMS mit einer Meldung quittiert, die zum Inhalt hat, dass kein Primärschlüsselwert in T_Kunde passend zum Fremdschlüsselwert 577 in F_KundenNr gefunden wurde.

D01-1

Fügen Sie den Kunden „Klaus Petersen“ in mit der Kundennummer 500 in die Datenbank ein (Die anderen Attributwerte können Sie sich ausdenken).

D01-2

Fügen Sie für Herrn Petersen ein Girokonto mit folgenden Angaben ein:

Kontoinhaber: „Karla Petersen“
Kontonummer: 555666777
Bankleitzahl: 10050000.

D01-3

Erfassen Sie den Artikel „Eiernudeln“ mit der Artikelnummer L007 und dem Mehrwertsteuersatz 8% (Die anderen Attributwerte können Sie sich ausdenken).

2. Daten ändern

Mit der Anweisung UPDATE werden Attributwerte in einer Relation aktualisiert. Einem oder mehreren Attributen wird ein neuer Wert zugewiesen. Dabei wird in einer WHERE-Klausel festgelegt, welche Tupel von der Änderung betroffen sein sollen. Ohne eine solche Angabe werden alle Zeilen der Tabelle geändert!

UPDATE-Anweisung

Die Grundform der Anweisung lautet:

```
UPDATE relation
    SET attributangabe = wertangabe { ,attributangabe = wertangabe}
    [WHERE bedingung]
```

Beispiele

Für den Kunden mit der Kundennummer 100 soll der Status in „G“ geändert werden.

```
UPDATE T_Kunden
    SET status = 'G'
    WHERE Kunden_Nr = 100;
```

Die Wertzuweisung an das Attribut „Status“ erfolgt in diesem Beispiel über eine Konstante. Die WHERE-Klausel gibt einen Vergleichswert 100 für „Kunden_Nr an. Infolgedessen werden alle Kundeneinträge, deren Kundennummer 100 ist, der Änderung unterzogen. Wegen der Primärschlüsseleigenschaft dieses Attributs ist dies nur ein einziger Kunde (oder gar keiner).

Im folgenden Beispiel wird bei allen Artikeln, deren Artikelnummer mit „K“ beginnt, der Listenpreis um 10% erhöht.

```
UPDATE T_Artikel
    SET Listenpreis = Listenpreis * 1.1
    WHERE Artikel_Nr LIKE 'K%';
```

Verarbeitung der UPDATE-Anweisung

Zunächst wird die WHERE-Klausel geprüft. Für jedes Tupel, das die Selektionsbedingung erfüllt, werden alle Ausdrücke in der SET-Klausel rechts vom Gleichheitszeichen auf Basis der alten Attributwerte berechnet. Anschließend wird den entsprechenden Attributen links vom Gleichheitszeichen jeweils das Ergebnis der Berechungen zugewiesen. Dies ist von besonderer Bedeutung, wenn von einer UPDATE-Anweisung mehrere Spalten betroffen sind.

Beispiel:

```
UPDATE T_Artikel
    SET Bestand = Bestand + 100, Mindestbestand = Bestand;
```

In der letzten Zeile wird für den Mindestbestand der alte Bestandswert - nicht der um 100 erhöhte - eingesetzt.

Wenn eine Kundennummer geändert wird – was man wegen weit reichender organisatorischer Konsequenzen lieber nicht tun sollte – wird nach den bei der Datendefinition festgelegten Regeln (referentielle Integrität → Aktualisierungsweitergabe) die Änderung an die Relationen T_Bestellung und T_Girokonten weitergegeben.

Beispiel

```
UPDATE T_Kunden
    SET Kunden_Nr = 888
    WHERE Kunden_Nr = 101;
```

Überprüfen Sie beim Testen dieser Anweisung, ob die Änderungen der Kundennummer auch tatsächlich in den Relationen T_Bestellungen und T_Girokonten übernommen werden.

Falls dies nicht der Fall sein sollte, begründen Sie warum dies nicht geschieht (Internetrecherche).

Im folgenden Beispiel wird der Umzug eines Kunden in der Datenbank nachvollzogen. Dabei ändern sich mehrere Attribute.

```
UPDATE T_Kunden
    SET Strasse = 'Am Beckersberg 1', PLZ = '24558', Ort = 'Ulzburg'
    WHERE Kunden_Nr = 103;
```

D02-1

Ändern Sie den Nachnamen des Kunden mit Kundennummer 102 auf „Hansen“. Lassen Sie sich vorher und nachher alle Kunden anzeigen.

D02-2

Lassen Sie alle Artikeldaten anzeigen. Erhöhen Sie dann in der Artikeltabelle alle Mindestbestände um 5. Lassen Sie nochmals die Artikeldaten anzeigen.

D02-3

Für den Artikel mit der Nummer 'K003' sollen der Listenpreis um 1,50 € und der Bestand um 10 erhöht werden.

D02-4

Verringern Sie den Kaufpreis aller in der Bestellung 960151 enthaltenen Artikel um 20%.

---

3. Daten löschen

Die letzte der drei grundlegenden Manipulationen von Relationen ist das Löschen von Tupeln.

DELETE-Anweisung

Die Grundform der Anweisung lautet:

```
DELETE
    FROM relation
    [WHERE bedingung]
```

Damit werden alle Tupel in der angegebenen Relation gelöscht, auf die die mit der WHERE-Klausel formulierte Bedingung zutrifft. Ohne Argument in der WHERE-Klausel löscht DELETE alle Tupel!

Die folgende Anweisung löscht den zuvor neu angelegten Kunden: ,.

```
DELETE
    FROM T_Kunden
    WHERE Kunden_Nr = 102;
```

In der Relation T_Kunden muss der entsprechende Kunde nun fehlen.

Wirkungen von Integritätsregeln beim Löschen

Wegen der referenziellen Integrität darf kein Kunde gelöscht werden, der etwas bestellt hat. Nach

```
DELETE
    FROM T_Kunden
    WHERE Kunden_Nr = 103;
```

muss das DBMS mit einer Fehlermeldung antworten, da für diesen Kunden Bestellungen existieren.

D03-X (mehrere Teilaufgaben)

Löschen Sie alle Bestellungen vom Kunden 103. Löschen Sie anschließend den Kunden 103. Überprüfen Sie, ob über die Löschweitergaben die Positionen der Bestellungen und das Girokonto des Kunden gelöscht wurden!