3 INSERT, UPDATE, DELETE

SQL lernen
ISBN 3-8273-2025-9

Die Befehle INSERT, UPDATE und DELETE dienen dem Einfügen, Ändern und Löschen von Daten. Die Daten werden im Regelfall nicht per SQL-Statement eingegeben (das wäre etwas umständlich), sondern über eine entsprechende Datenbankapplikation. Wenn Sie nicht als Programmierer arbeiten, werden Sie nur in Ausnahmefällen in die Verlegenheit geraten, entsprechende SQL-Statements selbst formulieren zu müssen.

3.1 INSERT

Die vollständig abstrakte INSERT-Anweisung lautet wie folgt:

```
INSERT INTO tabellenname
       (spalte1, spalte2...)
   VALUES
       (wert1, wert2...)
```

Das wollen wir gleich verwenden, um die Tabelle *t_art* zu ergänzen. Verschaffen wir uns zunächst einmal mit *SELECT * FROM t_art* einen Überblick über die Tabelle. Anschließend fügen wir als achten Datensatz *Telefon bei Partner(in)* ein.

```
INSERT INTO t_art
       (nummer, bezeichnung)
   VALUES
       (8, "Telefon bei Partner(in)")
```

Wenn Sie sich nun mit *SELECT * FROM t_art* die Daten anzeigen lassen, dann werden Sie sehen, dass ein achter Datensatz vorhanden ist.

Auch hier werden sowohl einfache als auch doppelte Anführungszeichen akzeptiert.

Nun werden wir ein wenig auf den DELETE-Befehl vorgreifen: Damit wir nicht allzu viel „Müll“ in die Tabelle *t_art* schreiben, wollen wir die Daten nach jeder INSERT-Anweisung wieder löschen:

```
DELETE FROM t_art
    WHERE nummer = 8
```

Damit werden alle Datensätze aus der Tabelle *t_art* gelöscht, deren Spalte *nummer* den Wert acht aufweist.

Nun zurück zum SQL-Statement INSERT: Die Reihenfolge, in der wir die Werte angeben, ist beliebig, allerdings müssen Werte- und Spaltenliste zusammenpassen. Wir wollen nun zunächst die Bezeichnung und dann die Nummer eingeben:

```
INSERT INTO t_art
        (bezeichnung, nummer)
    VALUES
        ("Telefon bei Partner(in)", 8)
```

Wie Sie sehen, funktioniert auch das. Löschen Sie wieder den Eintrag.

Im Übrigen ist es gar nicht erforderlich, dass alle Spalten mit Werten belegt werden.

```
INSERT INTO t_art
        (nummer)
    VALUES
        (8)
```

Wie Sie sehen, wird hier für die Spalte *bezeichung* der Wert NULL eingefügt. Löschen Sie auch diesen Eintrag und versuchen Sie, nur eine Bezeichnung einzugeben:

```
INSERT INTO t_art
        (bezeichnung)
    VALUES
        ("Telefon bei Partner(in)")
```

Wie Sie sehen, wird dabei automatisch eine Nummer eingefügt. Hier sind die Zusammenhänge etwas verzwickter: Eigentlich wurde die Spalte *nummer* als NOT NULL definiert. Die Eingabe eines Datensatzes, dessen Feld *nummer* nicht belegt ist, müsste also zu einer Fehlermeldung führen.

Schauen wir uns also mit *IBConsole* die Metadaten der Tabelle an:

```
/* Domain definitions */
CREATE DOMAIN LN AS INTEGER NOT NULL;
CREATE DOMAIN ST AS VARCHAR(25) CHARACTER SET ISO8859_1 COLLATE DE_DE;
```

```
/* Table: T_ART, Owner: SYSDBA */

CREATE TABLE T_ART
(
  NUMMER                        LN,
  BEZEICHNUNG                   ST NOT NULL,
 UNIQUE (BEZEICHNUNG),
 PRIMARY KEY (NUMMER)
);
SET TERM ^ ;

/* Triggers only will work for SQL triggers */


CREATE TRIGGER TRIG_ART FOR T_ART
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
 IF(NEW.nummer IS NULL)
  THEN NEW.nummer = GEN_ID(g_art, 1);
END
 ^

COMMIT WORK ^
SET TERM ;^
```

Nun ist jedoch auch ein Generator und ein Trigger definiert, der immer vor einer INSERT-Aktion ausgelöst wird. Dieser Trigger prüft, ob in der Spalte *nummer* der Wert NULL vorhanden ist, und fügt gegebenenfalls einen neuen Generatorwert ein.

Mehr zu den Themen Tabellendefinition, Generatoren und Trigger in den folgenden Kapiteln.

Nun wieder zurück zu der Anweisung INSERT: Wenn Sie alle Felder mit Werten belegen und die Werteliste in der Reihenfolge der Spalten bei der Tabellendefinition übergeben, dann können Sie die Spaltenliste weglassen:

```
INSERT INTO t_art
    VALUES
        (8, "Telefon bei Partner(in)")
```

Übung 3.1 Fügen Sie in die Tabelle *t_kunde* Ihre Adresse ein, zeigen Sie die Adresse an und löschen Sie dann wieder den Datensatz. Beachten Sie, dass auch für die Tabelle *t_kunde* ein Trigger besteht, der automatisch Werte für die Spalte *nummer* einfügt.

Daten aus einer Unterabfrage

Bislang haben wir bei INSERT eine Werteliste mit Konstanten übergeben. Es ist aber auch möglich, die einzufügenden Werte von einer Unterabfrage zu übernehmen. Statt des Schlüsselwortes VALUES und der Werteliste wird dann die Unterabfrage eingefügt.

```
INSERT INTO t_kunde
        (vorname, nachname)
    SELECT vorname, nachname
        FROM t_kunde
        WHERE nummer = 10780
```

Diese Anweisung kopiert *vorname* und *nachname* von Datensatz 10.780 als weiteren Datensatz in die Tabelle.

Dabei kann die Tabelle, aus der die Unterabfrage ihre Daten bezieht, durchaus von der Tabelle abweichen, in welcher die INSERT-Anweisung die Daten einfügt.

```
INSERT INTO t_kunde
        (vorname, nachname)
    SELECT vorname, nachname
        FROM t_mitarbeiter
```

Diese Anweisung fügt alle (!) Mitarbeiter der Kundentabelle hinzu, beschränkt auf die Spalten *vorname* und *nachname*.

Um die letzten eingefügten Datensätze anzuzeigen, kann die folgende SELECT-Anweisung verwendet werden:

```
SELECT *
    FROM t_kunde
    WHERE nummer + 50 > (SELECT MAX(nummer)
        FROM t_kunde)
```

Zum Löschen dieser Datensätze verwendet man dann die folgende Anweisung:

```
DELETE FROM t_kunde
    WHERE nummer > 10780
```

Wie Sie an den letzten Anweisungen sehr deutlich sehen, arbeitet SQL mengenorientiert und nicht satzorientiert.

Übung 3.2 Fügen Sie wieder alle Mitarbeiter der Kundentabelle hinzu, diesmal aber mit vollständiger Adresse. Bei denjenigen Mitarbeitern, bei denen eine private Telefonnummer angegeben ist, soll diese in der Spalte *tel* gespeichert werden.

3.2 UPDATE

Die vollständig abstrakte UPDATE-Anweisung lautet wie folgt:

```
UPDATE tabellenname
    SET spalte1 = wert1, spalte2 = wert2, ...
    WHERE bedingung
```

Unsere Experimente wollen wir wieder an der Tabelle *t_art* vornehmen. Dabei gehe ich davon aus, dass der Datensatz mit *8 | Telefon bei Partner(in)* noch in der Tabelle steht. Wenn Sie ihn schon gelöscht haben, dann wissen Sie ja, wie Sie ihn wieder einfügen.

```
UPDATE t_art
    SET bezeichnung = "Test"
    WHERE nummer = 8
```

Diese Anweisung ändert nun den Wert in der Spalte *bezeichnung* auf *Test*.

Wie Sie bei der abstrakten UPDATE-Anweisung gesehen haben, kann man auch mehr als eine Spalte ändern:

```
UPDATE t_art
    SET nummer = 9,
        bezeichnung = "Telefon bei Partner(in)"
    WHERE nummer = 8
```

Hier wird die Spalte *bezeichnung* wieder auf *Telefon bei Parter(in)* gesetzt, der Wert in der Spalte *nummer* wird auf neun geändert.

Übung 3.3 Ändern Sie in der Tabelle *t_mitarbeiter* die Nachnamen aller Personen namens *Müller* in *Mueller* und machen Sie anschließend diese Änderung wieder rückgängig.

UPDATE ohne WHERE-Klausel

Die Verwendung einer WHERE-Klausel ist bei UPDATE-Anweisungen nicht zwingend, es werden dann aber alle Reihen der Tabelle geändert.

Damit wir bei unseren Experimenten unseren Datenbestand nicht ernsthaft beschädigen, wollen wir der Tabelle *t_art* eine neue Spalte hinzufügen:

```
ALTER TABLE t_art
    ADD murks VARCHAR(30)
```

Den Befehl ALTER TABLE werden wir später noch sehr genau besprechen. Hier sei so viel verraten, dass der Tabelle *t_art* eine neue Spalte namens *murks* hinzugefügt würde, welche 30 Zeichen aufnehmen kann.

Wenn Sie sich nun mit *SELECT * FROM t_art* die Tabelle ansehen, dann werden Sie feststellen, dass die *murks*-Spalte mit NULL-Werten gefüllt ist. Dies wollen wir sogleich ändern:

```
UPDATE t_art
    SET murks = "Test"
```

Nun haben alle Zellen der Spalte *murks* den Wert *Test*.

Übung 3.4 Ändern Sie die Spalte *murks* so, dass die ersten vier Einträge den Wert *Test* behalten und die zweiten vier Einträge auf NULL zurückgesetzt werden.

Nun wollen wir die Werte der Spalte *bezeichnung* in die Spalte *murks* kopieren:

```
UPDATE t_art
    SET murks = bezeichnung
```

Erfreulicherweise ist dabei noch nicht einmal eine Unterabfrage erforderlich.

Übung 3.5 Es sollen wieder die Werte der Spalte *bezeichnung* nach *murks* kopiert werden, es sind aber die Nummer, ein Doppelpunkt und ein Leerzeichen voranzustellen. (Der erste Eintrag würde dann lauten: *1: Durchwahl in Firma*.)

UPDATE mit Unterabfrage

In einer UPDATE-Anweisung sind auch Unterabfragen möglich. Sollen über solche Unterabfragen die neuen Werte ermittelt werden, dann sind sie so zu gestalten, dass sie jeweils nur einen Datensatz ermitteln:

```
UPDATE t_art a
    SET murks = (SELECT nachname
        FROM t_kunde k
        WHERE k.nummer = a.nummer + 1000)
```

Diese UPDATE-Anweisung fügt einige Nachnamen aus der Tabelle *t_kunde* in die Spalte *murks* ein, beginnend mit dem Datensatz 1.001.

Anschließend wollen wir uns von der Spalte *murks* wieder trennen:

```
ALTER TABLE t_art
    DROP murks
```

Auch die Anweisung ALTER TABLE werden wir noch genauer behandeln.

Unterabfragen sind nicht nur zur Ermittlung der neuen Werte erlaubt, sondern auch in der WHERE-Klausel. Um dies zu demonstrieren, soll in die Tabelle *t_mitarbeiter* eine Spalte *bemerkung* aufgenommen werden, in welcher angezeigt wird, ob der betreffende Mitarbeiter ein Handy hat.

Dazu wird zunächst die Tabelle *t_mitarbeiter* ergänzt:

```
ALTER TABLE t_mitarbeiter
    ADD bemerkung VARCHAR(50)
```

Nun wird bei den betreffenden Mitarbeitern in die Spalte *bemerkung* der Text *Über Handy erreichbar* aufgenommen:

```
UPDATE t_mitarbeiter m
    SET m.bemerkung = "Über Handy erreichbar"
    WHERE m.nummer IN (SELECT t.mitarbeiter
        FROM t_tele t
        WHERE t.art = 4)
```

Übung 3.6 Fügen Sie der Bemerkung *Über Handy erreichbar* noch die Handy-Nummer hinzu.

3.3 DELETE

Die vollständig abstrakte DELETE-Anweisung lautet wie folgt:

```
DELETE FROM tabellenname
    WHERE bedingung
```

Wir haben die DELETE-Anweisung schon verwendet, um neu eingefügte Datensätze wieder zu löschen:

```
DELETE FROM t_art
    WHERE nummer = 8
```

Selbstverständlich können Sie die WHERE-Klausel so formulieren, dass mehrere Datensätze gelöscht werden:

```
DELETE FROM t_art
    WHERE nummer > 10
```

In diesem Fall würden alle Datensätze gelöscht, deren Nummer größer als zehn ist.

Übung 3.7 Fügen Sie Ihren Vor- und Nachnamen in die Tabelle *t_mitarbeiter* ein und löschen Sie ihn dann wieder.

DELETE ohne WHERE-Klausel

Sie können auch eine DELETE-Anweisung ohne WHERE-Klausel formulieren – dann werden eben alle Datensätze gelöscht.

```
DELETE FROM t_art
```

Die Anweisung ist syntaktisch korrekt, die Ausführung wird aber trotzdem verweigert, weil ein Fremdschlüssel die Tabelle referenziert. Die einzigen Tabellen, die Sie vollständig löschen könnten, wären die Tabellen *t_tele* und *t_posten*. Nachdem das Löschen der etwa 200.000 Datensätze von *t_posten* zu lange dauern würde, trennen wir uns jetzt vom Inhalt der Tabelle *t_tele* (keine Sorge, wir machen das gleich wieder rückgängig):

```
DELETE FROM t_tele
```

Hin und wieder kommt es vor, dass man eine Tabelle versehentlich vollständig löscht. Solange Sie mit ISQL arbeiten, ist das nicht weiter schlimm: Mit TRANSACTIONS|ROLLBACK können Sie die komplette Transaktion, somit auch das Löschen der Tabelle, rückgängig machen.

Der Nachteil dabei: Sie machen dabei auch alle anderen Aktionen seit Beginn der Transaktion rückgängig, im ungünstigsten Fall seit dem Programmstart von ISQL.

DELETE mit Unterabfrage

Auch bei der DELETE-Anweisung sind Unterabfragen erlaubt. Dazu gleich ein praktisches Beispiel: Um die Telefonrechnung unserer Firma zu senken, sollen die Handy-Nummern aller Mitarbeiter gelöscht werden:

```
DELETE FROM t_tele
    WHERE art = (SELECT nummer
        FROM t_art
        WHERE bezeichnung = "Handy")
```

Machen Sie auch diese Anweisung mit ROLLBACK rückgängig.