VIEWS UND WEITERE TABELLEN-OPERATIONEN

9.1 Vereinbarung und Einsatz von Views

9.1.1 Einrichtung von Views

Die CREATE VIEW-Anweisung

Von Ausnahmen abgesehen ist es – wie wir zuvor erläutert haben – nicht sinnvoll, die aus einer Verbund-Bildung erhaltenen Tabellenzeilen in einer gesonderten Tabelle abspeichern zu lassen, da dadurch die Redundanz innerhalb der Datenbasis erhöht und die Aktualität der jeweiligen Daten gefährdet wird. Vielmehr sollte ein Zugriff auf die resultierenden Werte möglich sein, der auf derjenigen Tabellen-Struktur basiert, die sich aus der auszuführenden Verbund-Bildung ergibt. Als zusätzliche Forderung sollte die Möglichkeit bestehen, bestimmte Tabellen-Inhalte vor dem Einblick unbefugter Anwender abzuschirmen.

- Diese beiden Forderungen lassen sich dadurch erfüllen, dass *Views* eingerichtet werden, die eine durch Verbund-, Projektions- und Selektions-Bildungen ermöglichte Sicht auf einen ausgewählten Datenbestand der Datenbasis zulassen.

Ein View ist eine *virtuelle* Tabelle, deren Zeilen und Spalten durch eine oder mehrere Tabellen der Datenbasis bestimmt sind. Dabei handelt es sich entweder um den Ausschnitt einer Tabelle, bei der der Zugriff auf die restlichen Zeilen und Spalten verdeckt wird, oder um den Verbund (evtl. mit nachfolgender Projektion bzw. Selektion) von Tabellen, der in seiner Gesamtheit bzw. nur als Ausschnitt zur Verfügung gestellt wird.

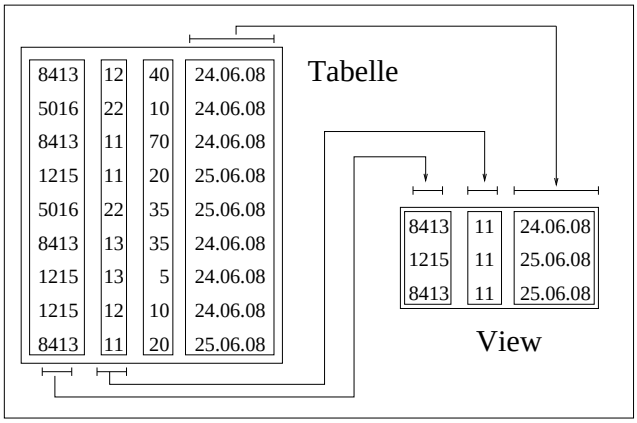

Abbildung 9.1: Beispiel für ein View

Von zentraler Bedeutung ist der Sachverhalt, dass ein View nur als Verweis auf Tabellenzeilen und Tabellenspalten, die physikalischer Bestandteil von bereits vorhandenen Tabellen der Datenbasis sind, zu verstehen ist und nicht als neue Form von physikalisch gespeicherten Datenbeständen.

Zum Beispiel kann man ein View, das auf der Tabelle UMSATZ basiert und nur alle Vertreternummern, Artikelnummern und Datumsangaben für die Umsätze der Artikel mit der

Nummer 11 enthält, einrichten lassen. Dieses View besitzt im Hinblick auf den Inhalt der Tabelle UMSATZ die innerhalb der Abbildung 9.1 angegebene Struktur.

Ein View enthält somit selbst keine Daten, sondern setzt sich aus Teilen einer oder mehrerer zuvor eingerichteter Tabellen (evtl. auch Views, siehe unten) zusammen. Es stellt somit eine Struktur-Beschreibung für den Aufbau einer Tabelle dar. Der zugehörige View-Inhalt lässt sich auf eine Anforderung hin – mittels einer SELECT-Anweisung – aus den Tabellenzeilen derjenigen Tabellen ermitteln, die als Ganzes oder als Tabellen-Ausschnitt innerhalb des Views einbezogen sind.

Views lassen sich wie folgt durch die CREATE VIEW-Anweisung vereinbaren:

```
CREATE [ OR REPLACE ] VIEW view-name
                [ ( spaltenname-1 [ , spaltenname-2 ]... ) ]
          AS SELECT-anweisung
```

Abbildung 9.2: Aufbau eines Views

Hinweis: Die hinter AS angegebene SELECT-Anweisung darf keine ORDER BY-Klausel enthalten.

Dadurch wird ein View namens “view-name” aufgebaut bzw. – beim Einsatz der Schlüsselwörter “OR REPLACE” – eine bereits vorhandene View-Vereinbarung ersetzt.

Voraussetzung zur Einrichtung eines Views ist, dass der Anwender das CREATE VIEW-Recht besitzt. Dieses Recht muss ihm durch den DB-Verwalter mittels der folgenden Anforderung zugeteilt worden sein:

```
GRANT CREATE VIEW TO gast;
```

Die Bildung von Viewnamen unterliegt denselben Regeln, die für den Aufbau von Tabellennamen gelten. Der Viewname darf zuvor noch nicht für eine Tabelle bzw. ein View innerhalb des aktuellen Schemas verwendet worden sein. Das resultierende View stellt sich als diejenige Tabellen-Struktur dar, die als Ergebnis der SELECT-Anweisung erhalten werden würde.

Grundlegend für das View sind die Tabellennamen, die in der SELECT-Anweisung innerhalb der FROM-Klausel angegeben werden. Anstelle von Tabellennamen dürfen dort auch Viewnamen aufgeführt sein. Dies bedeutet, dass Views – genauso wie Tabellen – als Bausteine für den Aufbau eines Views verwendet werden können.

Sollen für das View nicht die durch die SELECT-Anweisung spezifizierten Spaltennamen verabredet werden, so sind die gewünschten neuen Namen vor dem Schlüsselwort AS anzugeben. Die Zuordnung zu den Tabellenspalten erfolgt gemäß der Reihenfolge, in der diese Namen hintereinander aufgeführt sind. Die Angabe von Spaltennamen ist z.B. dann erforderlich, wenn Ausdrücke vor dem Schlüsselwort FROM innerhalb der SELECT-Anweisung angegeben sind.

Beispiele von Views

Im Hinblick auf die oben angegebene Darstellung lässt sich z.B. durch die Anweisung

```
CREATE VIEW UMSATZ_VIEW_1 AS
  SELECT V_NR, A_NR, DATUM FROM UMSATZ WHERE A_NR = 11;
```

das View UMSATZ_VIEW_1 einrichten, das auf der Tabelle UMSATZ basiert. Es enthält drei Spalten mit den Namen V_NR, A_NR und DATUM, die auf die korrespondierenden

Tabellenspalten der Tabelle UMSATZ weisen. Es werden aus der Basis-Tabelle UMSATZ die Tabellenspalte A_STUECK mit den Stückzahlen und ferner alle diejenigen Tabellenzeilen ausgeblendet, welche die Auswahl-Bedingung "A_NR = 11" nicht erfüllen. Folglich führt etwa die Anweisung

```
SELECT V_NR, A_NR FROM UMSATZ_VIEW_1;
```

zur folgenden Anzeige:

```
      V_NR       A_NR
---------- ----------
      1215         11
      8413         11
      8413         11
```

Wollen wir z.B. ein View einrichten, das durch den Verbund der Tabellen UMSATZ und ARTIKEL über die Tabellenspalten A_NR mit anschließender Projektion auf die Spalten A_NAME, A_STUECK und DATUM und ergänzender Spalte mit den Umsatzwerten aufgebaut werden soll, so können wir dazu die folgende CREATE VIEW-Anweisung eingeben:

```
CREATE VIEW UMSATZ_VIEW_2 (NAME,UMSATZ,STUECKZAHL,DATUM)
  AS SELECT A_NAME, A_PREIS*A_STUECK, A_STUECK, DATUM
            FROM UMSATZ INNER JOIN ARTIKEL USING (A_NR);
```

Dadurch wird das View UMSATZ_VIEW_2 mit den Spalten NAME, UMSATZ, STUECKZAHL und DATUM eingerichtet. Die aus der Projektion resultierenden Werte des Produkts "A_PREIS*A_STUECK" sind unter dem Spaltennamen UMSATZ zugänglich.

Da innerhalb einer SELECT-Anweisung hinter dem Schlüsselwort FROM nicht nur Tabellennamen, sondern auch View-Namen angegeben werden dürfen, lässt sich der Inhalt des Views UMSATZ_VIEW_2 anschließend durch

```
SELECT * FROM UMSATZ_VIEW_2;
```

wie folgt anzeigen:

```
NAME                      UMSATZ STUECKZAHL DATUM
-------------------- ---------- ---------- --------
Oberhemd                    1592         40 24.06.08
Mantel                      3600         10 24.06.08
Oberhemd                    3094         70 24.06.08
Oberhemd                     884         20 25.06.08
Mantel                     12600         35 25.06.08
Hose                      3867,5         35 24.06.08
Hose                       552,5          5 24.06.08
Oberhemd                    398         10 24.06.08
Oberhemd                    884         20 25.06.08
```

Wie oben angegeben, ist es erlaubt, bereits vorhandene Views in den Aufbau eines neuen Views einzubeziehen.
So können wir z.B. einen Ausschnitt aus den Werten des Views UMSATZ_VIEW_2 wie folgt festlegen:

```
CREATE VIEW UMSATZ_VIEW_3
       AS SELECT NAME, STUECKZAHL FROM UMSATZ_VIEW_2;
```

Dadurch ist das View UMSATZ_VIEW_3 innerhalb des aktuellen Schemas eingerichtet. Es enthält die Werte der Tabellenspalten A_NAME (der Tabelle ARTIKEL) und A_STUECK (der Tabelle UMSATZ). Somit kann indirekt – über den Einsatz des Views UMSATZ_VIEW_2 – auf den Datenbestand der Tabellen UMSATZ und ARTIKEL zugegriffen werden.

9.1.2 Löschung von Views

Views bleiben innerhalb einer Datenbasis solange vereinbart, bis sie durch die DROP VIEW-Anweisung in der folgenden Form gelöscht werden:

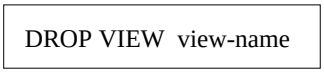

Abbildung 9.3: Löschung eines Views

Zum Beispiel können wir das oben aufgebaute View UMSATZ_VIEW_3 durch die folgende Anweisung aus dem Schema entfernen:

```
DROP VIEW UMSATZ_VIEW_3;
```

Auf ein View kann auch dann nicht mehr zugegriffen werden, wenn eine Tabelle oder ein View, auf das beim Aufbau des Views Bezug genommen wurde, innerhalb des Schemas entfernt worden ist.
Somit würde nach der Ausführung der Anweisung

```
DROP TABLE ARTIKEL;
```

bzw. der Anweisung

```
DROP VIEW UMSATZ_VIEW_2;
```

auf das oben aufgebaute View UMSATZ_VIEW_3 nicht mehr zugegriffen werden können.

9.1.3 Bestandsänderungen mittels eines Views

Sofern ein View nicht auf einem Verbund mehrerer Tabellen, sondern nur auf einer einzigen Tabelle basiert, besteht die Möglichkeit, über dieses View Veränderungen innerhalb der diesem View zugrunde liegenden Basis-Tabelle vorzunehmen. Dazu ist innerhalb diesbezüglicher INSERT-, UPDATE- bzw. DELETE-Anweisungen der Name des Views aufzuführen.

Somit können wir z.B. über das oben vereinbarte View UMSATZ_VIEW_1 durch die Anweisung

```
INSERT INTO UMSATZ_VIEW_1 VALUES (8413,11,'26.06.2008');
```

der Tabelle UMSATZ eine neue Tabellenzeile hinzufügen, in der für die Tabellenspalte A_STUECK der Nullwert eingetragen wird. Die anschließende Anforderung

```
SELECT * FROM UMSATZ WHERE A_NR = 11;
```

führt zur folgenden Anzeige:

```
      V_NR        A_NR   A_STUECK DATUM
---------- ---------- ---------- --------
      8413          11         70 24.06.08
      1215          11         20 25.06.08
      8413          11         20 25.06.08
      8413          11            26.06.08
```

Grundsätzlich werden in den Tabellenspalten der einem View zugrunde liegenden Tabelle – hier in der Spalte A_STUECK der Tabelle UMSATZ – dann Nullwerte eingetragen, wenn diese Spalten nicht Bestandteil des Views sind. Dies ist natürlich nur dann möglich, wenn die betreffenden Tabellenspalten bei der Vereinbarung der Tabellen nicht durch die Schlüsselwörter "NOT NULL" gekennzeichnet wurden, sodass nur von Nullwerten verschiedene Werte aufgenommen werden können.

Über ein View lassen sich auch Tabellenzeilen in eine Basis-Tabelle eintragen, die anschließend nicht Bestandteil des Views sind. So führt z.B. die folgende Anweisung zur Eingabe einer neuen Tabellenzeile in die Tabelle UMSATZ:

```
INSERT INTO UMSATZ_VIEW_1 VALUES (5016,12,'26.06.2008');
```

Die Anweisung

```
SELECT V_NR, A_NR, DATUM FROM UMSATZ
                         WHERE A_NR = 11 OR A_NR = 12;
```

liefert die folgende Anzeige:

```
      V_NR        A_NR DATUM
---------- ---------- --------
      1215          11 25.06.08
      1215          12 24.06.08
      5016          12 26.06.08
      8413          11 24.06.08
      8413          11 25.06.08
      8413          12 24.06.08
```

Hinweis: Es wird davon ausgegangen, dass die zuvor eingetragene Tabellenzeile mit dem Datumswert "26.06.2008" gelöscht wurde, sodass wieder der ursprüngliche Tabelleninhalt von UMSATZ vorliegt.

Dagegen wird durch die Anweisung

```
SELECT V_NR, A_NR, DATUM FROM UMSATZ_VIEW_1;
```

die Anzeige

```
      V_NR       A_NR DATUM
---------- ---------- --------
      1215         11 25.06.08
      8413         11 24.06.08
      8413         11 25.06.08
```

abgerufen, die das Fehlen der neuen Tabellenzeile von UMSATZ innerhalb des Views UMSATZ_VIEW_1 dokumentiert.

Es besteht die Möglichkeit, dass derartige Veränderungen der Basis-Tabelle, die nicht gleichzeitig durch das View erfasst werden, unterbunden werden können. Dazu sind innerhalb der CREATE VIEW-Anweisung, mit der das View vereinbart wird, die Schlüsselwörter "WITH CHECK OPTION" in der folgenden Form aufzuführen:

```
CREATE VIEW view-name
            [ ( spaltenname-1 [ , spaltenname-2 ]... ) ]
        AS SELECT-anweisung
        WITH CHECK OPTION
```

Abbildung 9.4: Aufbau eines Views mit Integritätsprüfung

Dies bedeutet, dass eine durch eine INSERT-, eine DELETE- oder eine UPDATE-Anweisung durchzuführende Änderung an einer Basis-Tabelle nur dann vorgenommen werden kann, wenn für die jeweils betroffene Tabellenzeile die in der WHERE-Klausel der View-Definition aufgeführte Auswahl-Bedingung zutrifft. Ist diese Bedingung nicht erfüllt, so wird die geforderte Änderung der Basis-Tabelle abgewiesen.

Haben wir ein View namens UMSATZ_VIEW_4 unter Einsatz der Schlüsselwörter "WITH CHECK OPTION" durch die Anweisung

```
CREATE VIEW UMSATZ_VIEW_4
           AS SELECT V_NR, A_NR, DATUM FROM UMSATZ
                                       WHERE A_NR = 11
              WITH CHECK OPTION;
```

definiert, so wird z.B. die durch die INSERT-Anweisung

```
INSERT INTO UMSATZ_VIEW_4 VALUES (5016,12,'26.06.2008');
```

geforderte Bestandsänderung der Tabelle UMSATZ abgewiesen.

Unabhängig von einer derartigen Sicherung werden auch alle Anforderungen zur Änderung eines Views zurückgewiesen, die sich auf eine Spalte beziehen, die sich durch eine arithmetische Operation aus den Werten der Basis-Tabelle ableitet.

Somit ist z.B. eine Änderung des durch die Anweisung

```
CREATE VIEW UMSATZ_ARTIKEL_VIEW (V_NR, A_NR, A_UMSATZ)
          AS SELECT V_NR, A_NR, A_STUECK*A_PREIS
                                FROM ARTIKEL_UMSATZ;
```

vereinbarten Views innerhalb der Spalte A_UMSATZ nicht möglich.

9.1.4 Bestandsänderungen mittels Einsatz von Instead-of-Triggern

Im Abschnitt 8.2 (im Anschluss an die Abbildung 8.10) haben wir erläutert, wie wir uns – auf der Basis der beiden Tabellen ARTIKEL und UMSATZ – durch die Verbund-Bildung

```
SELECT V_NR, UMSATZ.A_NR, A_NAME, A_PREIS,
       A_STUECK, DATUM
  FROM ARTIKEL, UMSATZ WHERE ARTIKEL.A_NR = UMSATZ.A_NR;
```

den bislang getätigten Umsatz anzeigen lassen können. Falls wir diese SELECT-Anweisung als Basis einer View-Vereinbarung verwenden und daher die Anweisung

```
CREATE OR REPLACE VIEW ARTIKEL_UMSATZ_V
          (V_NR,A_NR,A_NAME,A_PREIS,A_STUECK,DATUM)
    AS SELECT V_NR, UMSATZ.A_NR, A_NAME,
              A_PREIS, A_STUECK, DATUM
              FROM ARTIKEL, UMSATZ
              WHERE ARTIKEL.A_NR = UMSATZ.A_NR;
```

zur Ausführung bringen, lassen sich die Umsatzdaten anschließend wie folgt anzeigen:

```
SELECT * FROM ARTIKEL_UMSATZ_V;
```

Sofern die Eingabe neuer Umsatzdaten unmittelbar in Verbindung mit dem Neueintrag eines Artikels stehen soll, lässt sich das vereinbarte View ARTIKEL_UMSATZ_V *nicht* verwenden. Wie wir im vorausgegangenen Abschnitt 9.1.3 dargestellt haben, können in ein View nur dann Eingaben vorgenommen werden, wenn das View auf einer *einzigen* Tabelle basiert.

Um jedoch trotzdem – unter Einsatz des Views ARTIKEL_UMSATZ_V – die gewünschten Operationen durchführen zu können, lässt sich eine besondere Form eines Triggers einsetzen.

- Für Views – aber nicht für Tabellen – können *Instead-of-Trigger* verabredet werden. Falls ein derartiger Trigger feuert, können beliebige INSERT-, UPDATE- oder DELETE-Anweisungen für beliebige Tabellen zur Ausführung gebracht werden.

Hinweis: Es ist verboten, für ein View einen normalen Trigger (siehe Abschnitt 6.4) zu verabreden.

Um einen Instead-of-Trigger zu vereinbaren, ist die CREATE TRIGGER-Anweisung mit der INSTEAD OF-Klausel in der innerhalb der Abbildung 9.5 dargestellten Form einzusetzen.

Hinweis: Bei der Vereinbarung eines Instead-of-Triggers ist keine WHEN-Klausel und auch keine OF-Klausel hinter UPDATE erlaubt.

Ein in dieser Form vereinbarter Instead-of-Trigger feuert in dem Augenblick, in dem der angegebene Viewname innerhalb einer DELETE-, INSERT- bzw. UPDATE-Anweisung aufgeführt wird – je nachdem, welches Schlüsselwort bei der Trigger-Definition angegeben wurde. Es gelangen diejenigen Anweisungen zur Ausführung, die bei der Vereinbarung des Instead-of-Triggers festgelegt wurden.

```
CREATE [ OR REPLACE ] TRIGGER trigger-name
      INSTEAD OF { DELETE | INSERT | UPDATE }
      ON view-name
  BEGIN
      anweisung-1;
    [ anweisung-2; ]...
  END;
/
```

Abbildung 9.5: Vereinbarung eines Instead-of-Triggers

Um unsere oben angegebene Anforderung erfüllen zu können, lässt sich z.B. ein Instead-of-Trigger namens ARTIKEL_UMSATZ_V_TRIGGER wie folgt verabreden:

```
CREATE OR REPLACE TRIGGER ARTIKEL_UMSATZ_V_TRIGGER
          INSTEAD OF INSERT ON ARTIKEL_UMSATZ_V
BEGIN
  INSERT INTO ARTIKEL
             VALUES(:NEW.A_NR,:NEW.A_NAME,:NEW.A_PREIS);
  INSERT INTO UMSATZ
    VALUES(:NEW.V_NR,:NEW.A_NR,:NEW.A_STUECK,:NEW.DATUM);
END;
/
```

Hinweis: Zur Bedeutung von ":NEW" und ":OLD" siehe die Darstellung im Abschnitt 6.4.

Wird anschließend z.B. die Anforderung

```
INSERT INTO ARTIKEL_UMSATZ_V
       VALUES(8413,31,'Jacke',99.50,12,'26.06.2008');
```

gestellt, so feuert der Instead-of-Trigger ARTIKEL_UMSATZ_V_TRIGGER. Dies hat die implizite Ausführung von

```
INSERT INTO ARTIKEL VALUES(31,'Jacke',99.50);
```

zur Folge, sodass innerhalb der Tabelle ARTIKEL eine neue Zeile im Artikelbestand eingetragen wird. Zusätzlich wird der getätigte Umsatz in Form einer neuen Tabellenzeile innerhalb der Tabelle UMSATZ ergänzt, weil durch das Feuern des Triggers zusätzlich die Ausführung der folgenden Anweisung veranlasst wird:

```
INSERT INTO UMSATZ VALUES(8413,31,12,'26.06.2008');
```

- Grundsätzlich lässt sich ein Instead-of-Trigger für ein beliebiges View verabreden. Dabei dürfen auch SQL-Anweisungen zur Bestandsänderung von beliebigen Tabellen aufgeführt werden. Der Einsatz eines Instead-of-Triggers ist somit nicht eingeschränkt auf diejenigen Tabellen, die die Basis für den Aufbau des Views bilden.

Zum Beispiel können wir die oben angegebene Trigger-Vereinbarung dahingehend erweitern, dass zusätzlich zu den Ergänzungen in den Tabellen ARTIKEL und UMSATZ auch eine weitere Tabellenzeile in die Protokoll-Tabelle UMSATZ_PROTOKOLL (siehe Abschnitt 6.4) eingetragen wird. Um dies zu erreichen, lässt sich auf der Basis der Tabellen-Vereinbarung

```
CREATE TABLE UMSATZ_PROTOKOLL
             (ANWENDER CHAR(20), DATUM DATE,
              V_NR NUMBER(4), A_NR NUMBER(2));
```

z.B. die folgende Verabredung treffen:

```
CREATE OR REPLACE TRIGGER ARTIKEL_UMSATZ_V_TRIGGER
                  INSTEAD OF INSERT ON ARTIKEL_UMSATZ_V
BEGIN
  INSERT INTO ARTIKEL
         VALUES(:NEW.A_NR,:NEW.A_NAME,:NEW.A_PREIS);
  INSERT INTO UMSATZ
         VALUES(:NEW.V_NR,:NEW.A_NR,
                          :NEW.A_STUECK,:NEW.DATUM);
  INSERT INTO UMSATZ_PROTOKOLL
         VALUES(USER,SYSDATE,:NEW.V_NR,:NEW.A_NR);
 END;
/
```

Wir stellen uns jetzt die Aufgabe, das View ARTIKEL_UMSATZ_V dazu einzusetzen, innerhalb der Tabellen ARTIKEL und UMSATZ alle Zeilen mit einer vorgegebenen Artikelnummer zu löschen und gleichzeitig einen Eintrag über jede gelöschte Zeile innerhalb der Tabelle UMSATZ_PROTOKOLL vorzunehmen.
Um diese Aufgabe zu lösen, vereinbaren wir wie folgt einen Instead-of-Trigger namens ARTIKEL_UMSATZ_V_LOESCHEN_TRIG:

```
CREATE OR REPLACE TRIGGER ARTIKEL_UMSATZ_V_LOESCHEN_TRIG
                  INSTEAD OF DELETE ON ARTIKEL_UMSATZ_V
 BEGIN
   DELETE FROM UMSATZ WHERE A_NR=:OLD.A_NR;
   DELETE FROM ARTIKEL WHERE A_NR=:OLD.A_NR;
   INSERT INTO UMSATZ_PROTOKOLL
          VALUES(USER,SYSDATE,:OLD.V_NR,:OLD.A_NR);
 END;
/
```

Auf dieser Basis lassen sich z.B. alle Einträge für den Artikel mit der Kennzahl 31 innerhalb der Tabellen ARTIKEL und UMSATZ dadurch löschen, dass wir die Ausführung der folgenden Anweisung veranlassen:

```
DELETE FROM ARTIKEL_UMSATZ_V WHERE A_NR=31;
```