Oracle SQL – Data Manipulation Language

Stephan Karrer

Datenmanipulation (DML)

- Zeilen hinzufügen, Werte verändern, Zeilen löschen
- Data Manipulation Language Statements:
  - INSERT,
  - UPDATE,
  - DELETE,
  - MERGE
  - INSERT ALL/FIRST

INSERT: Neue Zeilen einfügen

```
INSERT  INTO departments (department_id, manager_id,
                          location_id, department_name)
        VALUES ( 70, 100, 1700, 'Public Relations' );


INSERT  INTO departments (department_name, department_id)
        VALUES ('Public Relations', 1);
```

- Bei expliziter Angabe der zu befüllenden Spalten müssen nur die Spalten in beliebiger Reihenfolge angeben werden, die einen Wert bekommen müssen. Alles andere ist optional.
- Selbstverständlich müssen die Werte zu den Spalten-Definitionen passen und die vorhandenen Einschränkungen berücksichtigen.

INSERT: Ohne Spaltenliste

```
INSERT  INTO departments
        VALUES ( 70, 'Public Relations', NULL, NULL);


INSERT INTO departments
       VALUES (280, 'Recreation', DEFAULT, 1700);
```

- Ohne Angabe der Spalten müssen alle schreibbaren Spalten in der korrekten Reihenfolge befüllt werden.
- Es stehen NULL und DEFAULT zur Verfügung.

INSERT: Zeilen aus vorhandenen Tabellen kopieren

```
INSERT  INTO my_employees
        SELECT  *  FROM employees;

INSERT  INTO my_departments (id, name)
        SELECT department_id, department_name
                FROM departments WHERE manager_id = 100;
```

- Selbstverständlich können die Werte auch per Abfrage geliefert werden.

INSERT: Mehrere Value-Tupel

```
INSERT INTO departments (department_name, department_id)
    WITH deps AS (
        SELECT 'Public Relations', 1 FROM dual
        UNION ALL
        SELECT 'Legal', 2 FROM dual
        UNION ALL
        SELECT 'Sales', 3 FROM dual
    )
    SELECT * FROM deps;
```

- Leider erlaubt Oracle nicht mehrere Tupel hinter der VALUES-Klausel, aber man kann als Alternative eine Unterabfrage verwenden.

INSERT: Zieltabelle mittels Unterabfrage definieren

```
INSERT INTO
    (SELECT department_name, department_id, location_id
       FROM departments)
  VALUES ('Public Relations', 1, 1700);


INSERT  INTO
     (SELECT department_name, department_id, location_id
        FROM departments WHERE location_id = 1700
        WITH CHECK OPTION )
  VALUES ('Public Relations', 1, 1700);
```

- Die Unterabfrage definiert die Spalten der Zieltabelle und wir schreiben auf die Basistabelle. Das geht nur für sog. schreibbare Views.
- WITH CHECK OPTION verhindert das Einfügen von Zeilen, die die WHERE-Klausel verletzen. Dadurch können zusätzliche Bedingungen für den INSERT angegeben werden. (siehe auch Infos zu Views)

UPDATE: Vorhandene Zeilen ändern

```
UPDATE  employees
        SET department_id = 70, manager_id = 100
        WHERE employee_id = 137 ;

UPDATE  my_employees
        SET last_name = UPPER(first_Name) ;
```

- Es können mehrere Spaltenwerte in einem Schritt aktualisiert werden.
- Fehlt die WHERE-Klausel werden alle Zeilen aktualisiert !
- Selbstverständlich müssen die Typen und Constraints berücksichtigt werden.
- Es können komplexe Ausdrücke bei der Zuweisung verwendet werden.
- Berechnete Spalten können nicht aktualisiert werden.

INSERT und UPDATE: Verwendung von DEFAULT-Werten

```
INSERT INTO departments
       VALUES (280, 'Recreation', DEFAULT, 1700);

UPDATE  departments
        SET manager_id = DEFAULT
        WHERE department_id = 20;
```

- Auch hier stehen NULL bzw. DEFAULT zur Verfügung.
- Falls kein DEFAULT-Wert definiert ist, wird die Spalte mit NULL initialisiert (sofern das erlaubt ist).

UPDATE: Spalten mit Unterabfragen aktualisieren

```
UPDATE  employees
        SET  job_id = (SELECT job_id FROM employees
                               WHERE employee_id = 207),
             department_id = (SELECT department_id
                                FROM departments
                                WHERE department_name = 'IT')
        WHERE employee_id = 67;
```

- Selbstverständlich können die Werte auch per Unterabfrage geliefert werden.
- Vorsicht:
  Sollte die Unterabfrage keinen Wert liefern, wird der fehlende Wert mit NULL initialisiert, sofern das erlaubt ist.
- Auch hier können mittels WHERE-Klausel viele Zeilen aktualisiert werden.

UPDATE: Mehrere Spalten mit einer Unterabfrage aktualisieren

```
UPDATE  employees
    SET  (job_id, department_id) =
        (SELECT job_id, department_id FROM employees
                        WHERE employee_id = 207)
    WHERE employee_id = 67;
```

- Sofern mehrere Spalten betroffen sind, kann auch eine Unterabfrage alle benötigten Werte liefern. Nur dann kann bei der Zuweisung die Tupel-Schreibweise benutzt werden.
- Selbstverständlich muss die Unterabfrage in Anzahl und Datentypen korrespondieren und genau eine Zeile liefern.
- Auch hier gilt:
  Sollte die Unterabfrage keinen Wert liefern, wird der fehlende Wert mit NULL initialisiert, sofern das erlaubt ist.

UPDATE: Komplexes Beispiel aus der Doku (12c) – Teil 1

```
UPDATE employees a
    SET department_id =
        (SELECT department_id FROM departments
           WHERE location_id = '2100'),
        (salary, commission_pct) =
        (SELECT 1.1*AVG(salary), 1.5*AVG(commission_pct)
          FROM employees b
          WHERE a.department_id = b.department_id)
    WHERE department_id IN
        (SELECT department_id FROM departments
          WHERE location_id = 2900
              OR location_id = 2700);
```

UPDATE: Komplexes Beispiel aus der Doku (12c) – Teil 2

- Die verschiedenen Formen der SET-Klausel können auch kombiniert werden.
- Die Unterabfrage kann auch korreliert sein, sofern sie genau eine Zeile je SET-Operation liefert.
- Auch in der WHERE-Klausel von DML-Anweisungen darf eine Unterabfrage verwendet werden.

- Was das Beispiel nicht zeigt:
  Analog zum INSERT kann auch beim UPDATE die Zieltabelle via Unterabfrage bzw. VIEW definiert sein (mit analogen Einschränkungen).

DELETE: Zeilen löschen

```
DELETE FROM employees  WHERE employee_id = 67;

DELETE FROM my_employee;

DELETE FROM employees
       WHERE department_id = (SELECT department_id
                              FROM departments
                              WHERE department_name =
                                   'Public Relations' );
```

- Die WHERE-Klausel spezifiert, welche Zeilen zu löschen sind.
  Fehlt diese, werden alle Zeilen gelöscht.
- Selbstverständlich werden eventuelle Constraints wie Fremdschlüssel-Beziehungen berücksichtigt.
- Analog zum INSERT kann auch beim DELETE die Zieltabelle via Unterabfrage bzw. VIEW definiert sein (mit analogen Einschränkungen).

DML: Fehlerprotokollierung

```
CREATE TABLE raises (emp_id NUMBER, sal NUMBER
   CONSTRAINT check_sal CHECK(sal > 8000));

-- Erzeugen der Fehlertabelle
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('raises', 'errlog');

INSERT INTO raises
   SELECT employee_id, salary*1.1 FROM employees
   WHERE commission_pct > .2
   LOG ERRORS INTO errlog ('my_bad') REJECT LIMIT 10;
```

- Fehler der DML-Anweisungen, nicht nur bei INSERT werden in der erstellten ERROR_LOG Tabelle protokolliert.
- Sofern das Limit bzgl. der Fehler verletzt wird, erfolgt ein Rollback der Anweisung.

DML: Verwendung der RETURNING-Klausel

```
VAR x1 NUMBER; VAR x2 VARCHAR2(12);  -- SQLPLus

INSERT INTO employees
      (employee_id, last_name, email, hire_date, job_id, salary)
 VALUES (999, 'Doe', 'x@example.com', SYSDATE, 'SH_CLERK', 2400)
 RETURNING employee_id, job_id  INTO :x1, :x2 ;

PRINT;  -- SQLPlus
```

- Durch die RETURNING-Klausel werden die aktualisierten Werte (INSERT, UPDATE) bzw. die gelöschten (DELETE) zurückgeliefert.
- Dadurch ist keine neue Abfrage, insbesondere bei berechneten Spalten nötig.
- Allerdings werden Variablen zur Aufnahme der Werte benötigt, was in SQL nur über sog. BIND-Variablen (SQLPlus) geht. In PL/SQL ist das kein Problem.
- Ab Version 23c ist die RETURNING-Klausel erweitert, um sowohl die alten als auch neuen Werte zu liefern.

MERGE: Bedingungsabhängiges Aktualisieren bzw. Einfügen

Syntax:

```
MERGE INTO  target_table [table_alias]
  USING (table|view|subquery) [alias]
  ON  (condition)
  WHEN MATCHED THEN
     UPDATE SET
         column1 = col_value1,
         column2 = col_value2,
         ...
     [ DELETE WHERE (where_condition) ]
  WHEN NOT MATCHED THEN
     INSERT (column_list)
     VALUES (column_values) ;
```

- Fasst INSERT/UPDATE/DELETE in einer Anweisung (Transaktion) zusammen.

MERGE: Ein Beispiel aus der Doku

```
MERGE INTO bonuses D
  USING (SELECT employee_id, salary, department_id
           FROM employees
           WHERE department_id = 80)  S
  ON (D.employee_id = S.employee_id)
  WHEN MATCHED THEN
     UPDATE SET D.bonus = D.bonus + S.salary * .01
     DELETE WHERE (S.salary > 8000)
  WHEN NOT MATCHED THEN
     INSERT (D.employee_id, D.bonus)
     VALUES (S.employee_id, S.salary * 0.1)
     WHERE  (S.salary <= 8000);
```

- Es muss mindestens eine der beiden WHEN-Klauseln angegeben werden.
- Es kann keine Spalte aktualisiert werden, die in der ON-Klausel referenziert wird.
- DEFAULT kann beim Setzen des Werts verwendet werden.
- Error-Logging-Klausel ist auch bei MERGE möglich.

INSERT-Anweisung für mehrere Tabellen (ab Version 9i)

```
INSERT ALL
  INTO sales (prod_id, cust_id, time_id, amount)
    VALUES (product_id, customer_id, weekly_start_date, sales_sun)
  INTO sales (prod_id, cust_id, time_id, amount)
    VALUES (product_id, customer_id, weekly_start_date+1, sales_mon)
  SELECT product_id, customer_id, weekly_start_date, sales_sun,
         sales_mon  FROM sales_input_table;
```

- Die Ergebnisse eine Unterabfrage können in mehrere Tabellen eingefügt werden.
- Jede gelieferte Zeile wird für jede INSERT-Anweisung betrachtet.
- Nicht jede Ziel-Tabelle muss alle gelieferten Spalten aufnehmen.
- Die einzufügenden Werte müssen nicht aus der Unterabfrage stammen und es können Ausdrücke zugewiesen werden (auch DEFAULT oder NULL)

INSERT-Anweisung für mehrere Tabellen

```
INSERT ALL

   INTO  sal_history VALUES (empid, hired, sal)
   INTO  mgr_history VALUES (empid, mgr, sal)

   SELECT  employee_id empid, hire_date hired, salary sal,
           manager_id mgr
     FROM employees WHERE department_id = 50;
```

- Auch hier darf die Angabe der Spaltenliste entfallen, sofern die Bedingungen wie bei der einfachen INSERT-Anweisung erfüllt sind.
- Falls alle Werte übernommen werden sollen, kann auch die Werte-Liste entfallen.
- INSERT ALL ist eine Anweisung (entweder komplett erfolgreich oder gar nicht).

INSERT-Anweisung für mehrere Tabellen mit Bedingung

```
INSERT ALL
  WHEN order_total < 1000000
    THEN INTO small_orders
  WHEN order_total > 1000000 AND order_total < 2000000
    THEN INTO medium_orders
  WHEN order_total > 2000000
    THEN INTO large_orders
  SELECT order_id, order_total, sales_rep_id, customer_id
    FROM orders;
```

- Für jede Ziel-Tabelle und jede angelieferte Zeile wird die WHEN-Klausel evaluiert. Die Bedingungen in der WHEN-Klausel müssen angelieferte Spaltenwerte referenzieren.
- Die Bedingungen müssen nicht disjunkt sein.
- Bei Bedarf kann sowohl die Spaltenliste als auch die Werteliste angegeben werden.
- Angelieferte Zeilen, die keine Bedingung erfüllen, werden nicht berücksichtigt.

INSERT-Anweisung für mehrere Tabellen mit ELSE-Zweig

```
INSERT ALL
  WHEN order_total < 1000000
    THEN INTO small_orders
  WHEN order_total > 1000000 AND order_total < 2000000
    THEN INTO medium_orders
  ELSE INTO large_orders
SELECT order_id, order_total, sales_rep_id, customer_id
    FROM orders;
```

- Angelieferte Zeilen, die keine WHEN-Klausel erfüllen, werden via ELSE in eine Ziel-Tabelle eingefügt.

INSERT FIRST -Anweisung mit Bedingung

```
INSERT FIRST
  WHEN ottl < 100000 THEN
     INTO small_orders VALUES(oid, ottl, sid, cid)
  WHEN ottl > 100000 and ottl < 200000 THEN
     INTO medium_orders VALUES(oid, ottl, sid, cid)
  WHEN ottl > 290000 THEN
     INTO special_orders VALUES(oid, ottl, sid, cid)

  ELSE INTO large_orders VALUES(oid, ottl, sid, cid)
  SELECT o.order_id oid, o.customer_id cid,
         o.order_total ottl, o.sales_rep_id sid
     FROM orders o ;
```

- Für die erste erfolgreiche WHEN-Klausel wird die entsprechende INTO-Klausel ausgeführt und alle nachfolgenden bedingten Klauseln übersprungen.
- Generell sind Spalten-Aliase für die angelieferten Spalten möglich.