Snowflake – 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

```
-- Test-Tabelle
CREATE TABLE my_departments ( id      DECIMAL(4,0),
                              name    VARCHAR(30) );

INSERT  INTO my_departments ( id, name)
       VALUES  ( 1 , 'Abteilung1'),
               ( 2 , 'Abteilung2'),
               ( 3 , 'Abteilung3');
```

- Mehrere Tupel hinter der VALUES-Klausel sind erlaubt.

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.
- Auch hier können mittels WHERE-Klausel viele Zeilen aktualisiert werden.

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.
- Analog zum INSERT kann auch beim DELETE die Zieltabelle via Unterabfrage bzw. VIEW definiert sein (mit analogen Einschränkungen).

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 ]
  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

```
CREATE TABLE target_table (ID INTEGER, description VARCHAR);

CREATE TABLE source_table (ID INTEGER, description VARCHAR);

INSERT INTO target_table (ID, description) VALUES
    (10, 'To be updated (this is the old value)')
    ;

INSERT INTO source_table (ID, description) VALUES
    (10, 'To be updated (this is the new value)')
    ;

MERGE INTO target_table USING source_table
    ON target_table.id = source_table.id
    WHEN MATCHED THEN
        UPDATE SET target_table.description =
source_table.description;
```

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.

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.