SQL Server DML Statements

This SQL Tutorial focuses on the SQL Server **DML statements**. DML (Data Manipulation Language) statements are the element in the SQL language that is used for data retrieval and manipulation. Using these statements you can perform operations such as: adding new rows, updating and deleting existing rows, merging tables and so on.

This SQL tutorial allows you to become familiar with the following topics :

- INSERT – adding new rows to a table
  - Basic Syntax
  - Using a Column List
  - INSERT INTO SELECT
- UPDATE – updating values of fields
- DELETE – deleting rows from a table
- MERGE – merging tables
- TRANSACTIONS – learn what transactions are and how they can be managed

Please note – the SQL Server SELECT statement, another DML component, is not covered in this tutorial. For more details about the SQL Server SELECT statement please use the following link.

SQL Server DML – INSERT Statement

The SQL Server INSERT statement is used to add new rows to a table. Before you become familiar with this statement, please take into account the following points:

- Default value – when creating a new table (DDL), it is possible to define a specific column with a default value. This means that unless otherwise indicated, this is the value that you would like to enter into the field of that column. For example: default salary is 5700, and default employment date is '2014-23-01'.
- The following INSERT demonstrations are based on the table below, where the default value 5700 is assigned to the emp_sal column and currently the table has a single row

emp_id	emp_name	emp_sal	emp_hiredate
1	John	5700	2013/01/23

Basic Syntax

```
INSERT INTO table_name
VALUES (value, value, value  …)
```

Inserting a new row into the table

```
INSERT INTO emps
VALUES (2 , 'David' , 3200 , '2014/03/20')
```

The following table reflects the data in Emps table after the INSERT statement has completed:

emp_id	emp_name	emp_sal	emp_hiredate
1	John	5700	2013/01/23

2	David	3200	2014/03/20

Important Points

- The **order of the values** must match the order of the columns in the table. In the case above, the first value must be associated with the employee number column, the second value must be associated with the employee name column, and so on.
- **The value type** must match the type of columns in the table. For example, it is not possible to enter the value 37 into the emp_hiredate column.
- **The number of values** – must match the number of values in the table. It is not possible to enter more or less values than the number of values in the table (in this case, 4 values).
- **To enter a NULL value** into one of the fields of one of the columns, the NULL keyword should be used.

```
1 INSERT INTO emps
2 VALUES (3 , NULL , 3200 , '2009/09/09')
```

- **To enter a DEFAULT value** into one of the fields of one of the columns, the DEFAULT keyword should be used.

```
1 INSERT INTO emps
2 VALUES (4 , 'Maya', DEFAULT , '2011/09/09')
```

- **GETDATE()** – you can use the GETDATE () function instead of specifying a date.

```
1 INSERT INTO emps
2 VALUES (4 , 'Anne', DEFAULT , GETDATE())
```

Using a Column List

The following example uses a column list to explicitly specify the values that are inserted into each column. This option provides us with additional flexibility.

```
INSERT INTO  table_name (column_name , column_name , cc
VALUES (value, value, value …)
```

Inserting a new row into the table.

```
INSERT INTO emps (emp_id , emp_name , emp_salary ,  emp
VALUES (5 , 'Roy' , 3200 , GETDATE())
```

- The order **of the values** must match the order of the columns in the column list. In the case above, the first value must be associated with the employee number column; the second value must be associated with the employee name column, and so on.
- **The value type** must match the type of columns in the column list. For example, it is not possible to enter the value 37 into the emp_hiredate column.
- **The number of values** – must match the number of values in the column list. Therefore, there is no need to enter a value into each column of the table.

```
INSERT INTO emps (emp_id , emp_name)
VALUES (6 , 'Roger')
```

A NULL value is entered into a column that was not specified in the column list, if no default value exists; otherwise, the DEFAULT value is entered.

- **To enter a NULL value** explicitly into one of the fields of one of the columns, the NULL value should be specified.

```
INSERT INTO emps (emp_id , emp_name, emp_salary)
VALUES (6 , 'Ben' , NULL)
```

- **To enter a DEFAULT value** explicitly into one of the fields of one of the columns, the DEFAULT value should be specified.

```
1 INSERT INTO emps (emp_id , emp_name, emp_salary)
2 VALUES (7, 'Kim' , DEFAULT)
```

- **GETDATE** – you can use the GETDATE() function instead of specifying a date.

SQL Server INSERT INTO SELECT

This method allows copying data items from another table into the requested target table.

```
1 INSERT INTO  target_table_name (column_name, column_nam
2 SELECT  …
3 FROM    source_table_name
4 WHERE   …
```

For example:

```
1 INSERT INTO emps (emp_id , emp_name , emp_salary)
2 SELECT employee_id , last_name , salary
3 FROM   employees
4 WHERE  department_id = 50
```

- You can use the INSERT INTO SELECT statement either with or without the column list
- It is not mandatory to include a WHERE statement in the query; however, including such a statement is advisable if you would like to avoid copying the entire table.

SQL Server DML – UPDATE Statement

The SQL Server UPDATE statement is used to modify existing rows.

```
1 UPDATE table_name
2 SET column=value, column=value ..
```

```
3 WHERE condition
```

Updating the salary of employee no. 100:

```
1 UPDATE employees
2 SET salary = 5000
3 WHERE employee_id = 100
```

It is possible to update several fields at the same time, for example update salary, last name and first name of employee no. 100:

```
1 UPDATE employees
2 SET salary = 5000 , last_name = 'Doe' ,
3     first_name = 'John'
4 WHERE employee_id = 100
```

A subquery can be nested in the SQL Server UPDATE statement. For example, updating the salary of the employees in department 60 so it would match the average salary of the employees in department 50:

```
1 UPDATE employees
2 SET salary = (SELECT AVG(salary)
3               FROM employees
4               WHERE department_id = 50)
5 WHERE department_id = 60
```

It is possible to update to a NULL or DEFAULT value.

```
1 UPDATE employees
2 SET salary = DEFAULT
3 WHERE last_name = 'King'
4
5 UPDATE employees
6 SET salary = NULL
7 WHERE department_id = 90
```

Executing the SQL Server UPDATE statement without using the SQL Server WHERE clause results in updating all of the fields in a column.

SQL Server DML – DELETE Statement

The SQL Server DELETE statement is used to remove existing rows from a table.

```
DELETE FROM table_name
WHERE condition
```

Deleting the row that contains the data of employee no. 103:

```
DELETE FROM employees
WHERE employee_id = 103
```

A subquery can be nested in the DELETE statement.

```
DELETE FROM employees
WHERE department_id = (SELECT department_id FROM depart
                       WHERE department_name = 'Sales')
```

- Executing the SQL Server DELETE statement without using the SQL Server WHERE clause, results in the deletion of all of the rows in the table.
- It is not possible to delete only a single field by using the SQL Server DELETE statement .The SQL Server DELETE statement deletes rows.
- You can write a SQL Server DELETE statement either with or without the SQL Server FROM keyword.

```
DELETE FROM employees
-- or
DELETE employees
```

SQL Server DML – MERGE Statement

The SQL Server MERGE statement is used to synchronise the data of two tables, based on differences found between them, if the same row exists in both tables (row with the same customer id for example), but still each row has different values (each table holds a different phone number of that customer), UPDATE operation will

be executed. If the row only exists in one table, INSERT operation will be executed.

The following illustrations demonstrates the merge concept. At the end of each month, the data of Customers table is merged with the data of the Backups table. In the beginning of the month, the only MERGE activity was transferring (INSERT) all rows, as is, to the Backup table (provided that this was the first time that the Backup table was populated).

Customers

Customer ID	Customer Name	Internet Package Plan
1	John	1.5
2	David	2.5
3	George	2.5
4	Anne	5

Merging for the first time, only INSERT is needed. →

Backup

Customer ID	Customer Name	Internet Package Plan
1	John	1.5
2	David	2.5
3	George	2.5
4	Anne	5

At the end of the month, due to the changes that were carried out in the Customers table, the data items are merged with the Backup table as follows:

- For customers that already exist in the backup table, an UPDATE operation is carried out, for example, George's Internet Package will be changed from 2.5 MB to 5.0 MB.
- For new customers, an INSERT operation is carried out, thereby adding a new customer – for example, Roy– to the backup table.

Customers

Customer ID	Customer Name	Internet Package Plan
1	John	1.5
2	David	2.5
3	George	5
4	Anne	5
5	Roy	10

Merging again, this time using INSERT and UPDATE →

Backup

Customer ID	Customer Name	Internet Package Plan
1	John	1.5
2	David	2.5
3	George	5
4	Anne	5
5	Roy	10

Basic Syntax

```
01 MERGE INTO destination_table alias
02 USING source_table alias
```

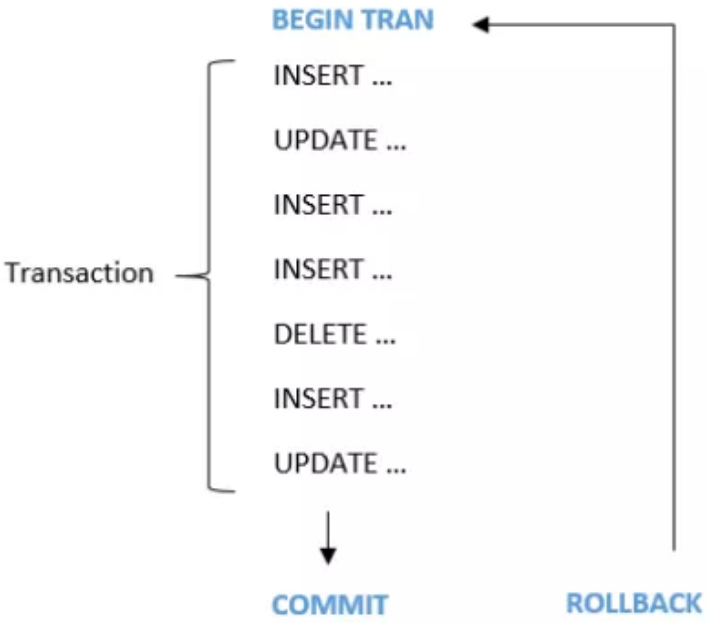
```
ON condition
WHEN MATCHED THEN
UPDATE SET
destination_table_alias.column = source_table_alias.co
destination_table_alias.column = source_table_alias.co
…
WHEN NOT MATCHED THEN
INSERT VALUES  (source_table_alias.column, source_tabl
               )
```

For example

```
MERGE INTO  customers_backup bkup
USING       customers cust
ON         (bkup.cust_id = cust.cust_id)
WHEN MATCHED THEN
UPDATE SET
bkup.cust_name = cust.cust_name ,
bkup.cust_surfing_package = cust.cust_surfing_package
WHEN NOT MATCHED THEN
INSERT VALUES(cust.cust_id , cust.cust_name , cust.cust
```

Database Transactions

Transactions are a single unit of various modification commands (such as UPDATE, INSERT, DELETE), which in most cases are associated with a single logical group. The term “a single logical group” refers to a set of operations with logical connection; for example: a batch of DML operations that are meant for updating specific data items in the customers table.



The COMMIT command used to save all changes made by the transaction in the database. The COMMIT command saves all modifications since the last COMMIT or ROLLBACK command.

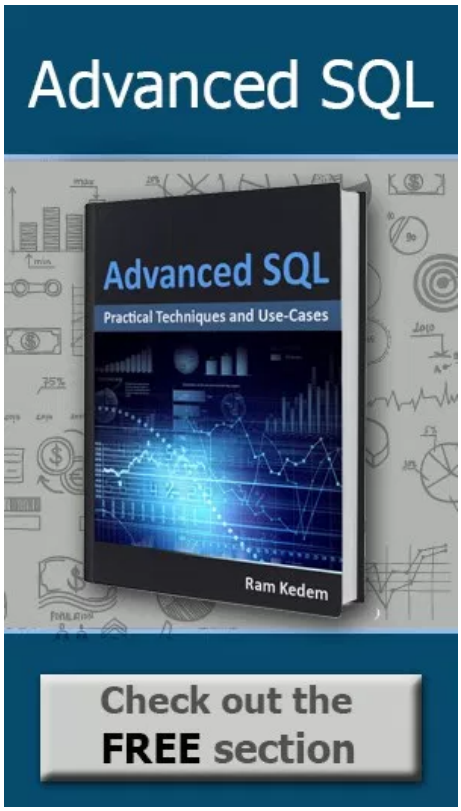
The ROLLBACK command used to undo changes made by a transaction. The ROLLBACK command can only undo modifications since the last COMMIT or ROLLBACK command that was issued.

SQL Server Autocommit Mode – In SQL Server, by default, every modification (such as UPDATE, DELETE, INSERT) is committed automatically once it completes. In SQL Server you can start an Explicit Transaction (one that you decide when and how to close) using the **BEGIN TRAN** command.

Share this:

Google Facebook LinkedIn Twitter Print Email

Login


Advanced SQL
Advanced SQL
Practical Techniques and Use-Cases
Ram Kedem
Check out the
FREE section