Merge

Author: Prabin Baniya with No comments

MERGE

Merge is a Data Manipulation Language that allows you to perform multiple DML operations in a very efficient manner. Some calls it UPSERT also. Merge reduces the multiple IF-statements and the costly full table scan for each "IF" statement. It took me few years before I started using Merge statement to my code. Once you start using it, you will not go back to **If-MESS** again.

Using MERGE statement to select rows from one or more sources for update, delete or insertions from a destination table or view. This will avoid multiple INSERT, UPDATE, and DELETE DML operations and makes less overhead to your database engine by less I/O.

The Merge Syntax consist of 5 primary clauses:
1) The **MERGE** clause specifies the table or view that is the target of the insert, update, or delete operations.
2) The **USING** clause specifies the data source being joined with the target.
3) The **ON** clause specifies the join conditions that determine where the target and source match.
4) The WHEN clauses (**WHEN MATCHED, WHEN NOT MATCHED BY TARGET, and WHEN NOT MATCHED BY SOURCE**) specify the actions to take based on the results of the ON clause and any additional search criteria specified in the WHEN clauses.
5) The **OUTPUT** clause returns a row for each row in the target that is inserted, updated, or deleted. This clause is optional, but useful to debug.

Sql Server :
Source Table: emp_src

```
CREATE TABLE [dbo].[Emp_Stage](
 [ID] [int] NOT NULL,
 [Name] [nvarchar](50) NOT NULL,
 [Salary] [int] NOT NULL,
 [Start_date] [nchar](10) NOT NULL,
 [City] [nvarchar](50) NOT NULL,
 [Region] [char](1) NOT NULL
) ON [PRIMARY]
```

```
--Insert test data.
insert into.Emp_Stage (ID, name,    salary, start_date, city,       region)
             values (1,  'Jason', 40420,  '02/01/94', 'New York', 'W');
insert into Emp_Stage (ID, name,    salary, start_date, city,       region)
            values (2,  'Robert',14420,  '01/02/95', 'Vancouver','N');
 insert into Emp_Stage (ID, name,    salary, start_date, city,       region)
             values (3,  'Celia', 24020,  '12/03/96', 'Toronto',  'W');
```

let's create Destination Table: **emp_dest:**
We will use Sql Server version of CTAS to create a table.

```
SELECT *
INTO   emp_dest
FROM  emp_stage
WHERE  id = 0;
-- Verify your destination table;
select * from emp_dest;
```

The merge statement below does two things:
1) If the records are in both source and destination table, the salary of destination table gets updated.
2) If the records does not exist in destination table, the missing record gets inserted to destination table and also update the salary from source to destination.

```
MERGE emp_dw AS D
using (SELECT id,
            name,
            salary,
            start_date,
            city,
            region
      FROM   emp_stage) AS S
ON D.id = s.id
WHEN matched THEN
  UPDATE SET D.salary = S.salary
WHEN NOT matched THEN
  INSERT (id,
          name,
          salary,
          start_date,
```

```
        city,
        region)
VALUES (S.id,
        S.name,
        S.salary,
        S.start_date,
        S.city,
        S.region);
```

Running the above command for the first time will copy everything from source table to destination. Adding a new record at source
table and re-running a merge will insert new record to destination table and also update salary column. You can perform the test by modifying source data like salary and/or by adding a new records. You must re-run Merge after source modifying a source data.

Additional Information:
It is important to understand how the source and target data are merged into a single input stream and how additional search criteria can be used to correctly filter out unneeded rows. Otherwise,you might specify the additional search criteria in a way that produces incorrect results.
Rows in the source are matched with rows in the target based on the join predicate specified in the ON clause. The result is a combined input stream. One insert, update, or delete operation is performed per input row. Depending on the WHEN clauses specified in the statement, the input row might be any one of the following:

1. A matched pair consisting of one row from the target and one from the source. This is the result of the WHEN MATCHED clause.
2. A row from the source that has no corresponding row in the target. This is the result of the WHEN NOT MATCHED BY TARGET clause.
3. A row from the target that has no corresponding row in the source. This is the result of the WHEN NOT MATCHED BY SOURCE clause.
4. **Syntax:**

```
-- MERGE statement with join conditions that produce unexpected results.
USE tempdb;
GO
BEGIN TRAN;
MERGE Target AS T
USING Source AS S
ON (T.EmployeeID = S.EmployeeID AND T.EmployeeName LIKE 'S%'
```

```
    AND S.EmployeeName LIKE 'S%' )
WHEN NOT MATCHED BY TARGET
    THEN INSERT(EmployeeID, EmployeeName) VALUES(S.EmployeeID, S.EmployeeName)
WHEN MATCHED
    THEN UPDATE SET T.EmployeeName = S.EmployeeName
WHEN NOT MATCHED BY SOURCE
    THEN DELETE
OUTPUT $action, Inserted.*, Deleted.*;
ROLLBACK TRAN;
GO
```

Oracle DB:

Good News To Oracle user, the same syntax works for Oracle.
Oracle Example from Oracle Doc:

```
CREATE TABLE bonuses (employee_id NUMBER, bonus NUMBER DEFAULT 100);
INSERT INTO bonuses(employee_id)
  (SELECT e.employee_id FROM employees e, orders o
  WHERE e.employee_id = o.sales_rep_id
  GROUP BY e.employee_id);
SELECT * FROM bonuses ORDER BY employee_id;

EMPLOYEE_ID      BONUS
----------- ----------
       153        100
       154        100
       155        100
       156        100
       158        100
       159        100
       160        100
       161        100
       163        100

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*.01)
    WHERE (S.salary <= 8000);

SELECT * FROM bonuses ORDER BY employee_id;
EMPLOYEE_ID      BONUS
----------- ----------
       153        180
       154        175
       155        170
       159        180
       160        175
       161        170
       179        620
       173        610
       165        680
       166        640
       164        720
       172        730
       167        620
       171        740
```

I ask you to go over your company legacy code and update that with MERGE where possible. The use of Merge makes your code optimized, faster, and more maintainable and that is what your client and your boss wants, correct?
If you are crazy and need to know more on MERGE, you can visit Oracle doc or Microsoft document page.