MySQL UPSERT

UPSERT is one of the essential features of DBMS software for **managing the database**. This operation allows the DML users to insert a new record or update existing data into a table. An UPSERT is made up of a combination of two words named **UPDATE** and **INSERT**. The first two letters, i.e., UP stands for UPDATE while the SERT stands for INSERT. The UPSERT is an atomic operation that means it is an operation that completes in a single-step. For example, if a record is new, it will trigger an INSERT command. But, if it already exists in the table, then this operation will perform an UPDATE statement.

By default, MySQL provides the ON DUPLICATE KEY UPDATE option to INSERT, which accomplishes this task. However, it also contains some other statements to fulfill this objective, such as INSERT IGNORE or REPLACE. We will learn and see all these solutions in detail.

MySQL UPSERT Example

We can perform MySQL UPSERT operation mainly in three ways, which are as follows:

1. UPSERT using INSERT IGNORE
2. UPSERT using REPLACE
3. UPSERT using INSERT ON DUPLICATE KEY UPDATE

UPSERT using INSERT IGNORE

INSERT IGNORE statement is used to ignore our execution errors when we perform an insert operation of invalid rows. For example, the primary key column cannot allow us to store duplicate values. If we try to insert a new record with the same primary key that already exists in the table, we will get an error. However, if we perform this action with the INSERT IGNORE command, it will generate a warning instead of an error.

Syntax

The following are syntax to use the **INSERT IGNORE** statement in MySQL:

```
INSERT IGNORE INTO table_name (column_names)
VALUES ( value_list), ( value_list) .....;
```

MySQL INSERT IGNORE Example

Let us understand the working of the INSERT IGNORE statement in MySQL. First, we need to create a table named **"Student"** using the following statement:

```
CREATE TABLE Student (
  Stud_ID int AUTO_INCREMENT PRIMARY KEY,
  Name varchar(45) DEFAULT NULL,
  Email varchar(45) NOT NULL UNIQUE,
  City varchar(25) DEFAULT NULL
);
```

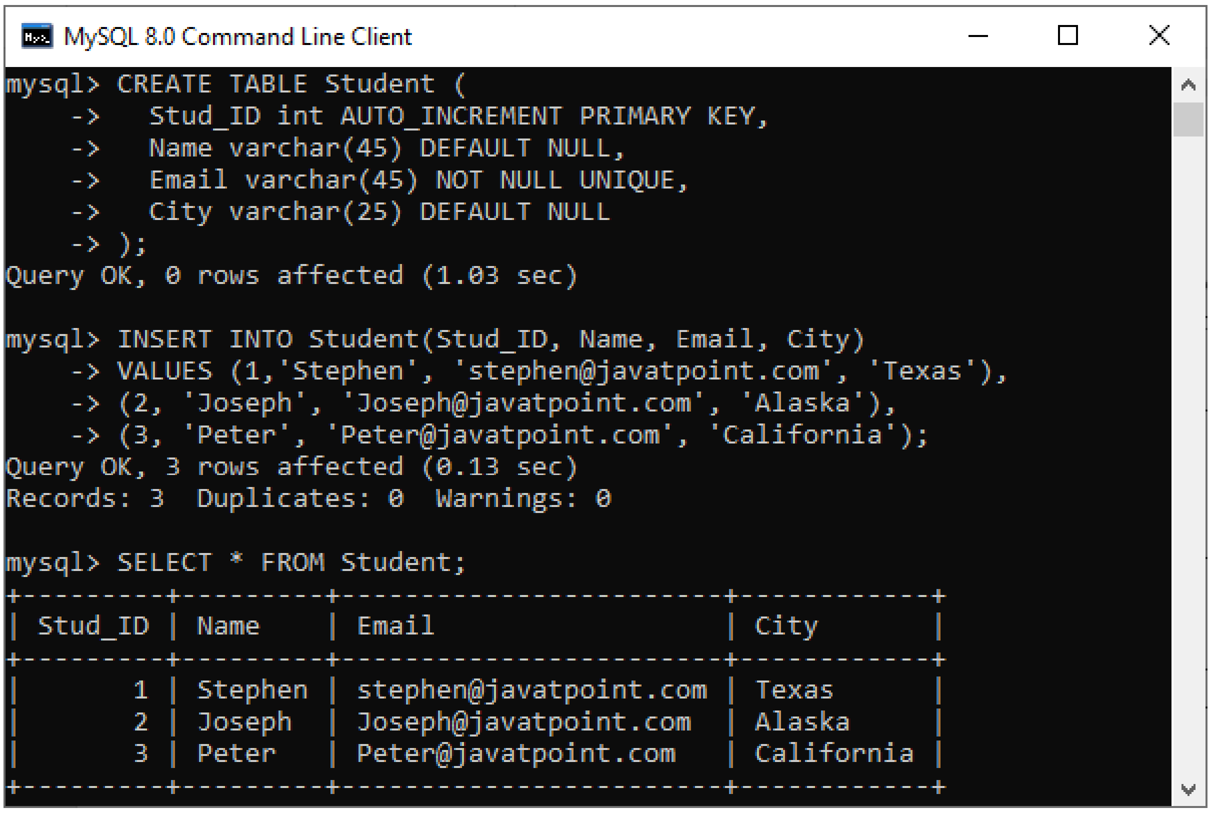
The **UNIQUE** constraint ensures that we cannot keep duplicate values into the **email column**. Next, it is required to insert the records into the table. The below statement is used to add data into a table:

```
INSERT INTO Student(Stud_ID, Name, Email, City)
VALUES (1,'Stephen', 'stephen@javatpoint.com', 'Texas'),
(2, 'Joseph', 'Joseph@javatpoint.com', 'Alaska'),
(3, 'Peter', 'Peter@javatpoint.com', 'California');
```

Now, we can verify the insert operation by using the **SELECT** statement:

1. mysql> **SELECT** * **FROM** Student;

We will get the below output where we have **three** rows into the table:



```
mysql> CREATE TABLE Student (
    ->   Stud_ID int AUTO_INCREMENT PRIMARY KEY,
    ->   Name varchar(45) DEFAULT NULL,
    ->   Email varchar(45) NOT NULL UNIQUE,
    ->   City varchar(25) DEFAULT NULL
    -> );
Query OK, 0 rows affected (1.03 sec)

mysql> INSERT INTO Student(Stud_ID, Name, Email, City)
    -> VALUES (1,'Stephen', 'stephen@javatpoint.com', 'Texas'),
    -> (2, 'Joseph', 'Joseph@javatpoint.com', 'Alaska'),
    -> (3, 'Peter', 'Peter@javatpoint.com', 'California');
Query OK, 3 rows affected (0.13 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Student;
+---------+---------+------------------------+------------+
| Stud_ID | Name    | Email                  | City       |
+---------+---------+------------------------+------------+
|       1 | Stephen | stephen@javatpoint.com | Texas      |
|       2 | Joseph  | Joseph@javatpoint.com  | Alaska     |
|       3 | Peter   | Peter@javatpoint.com   | California |
+---------+---------+------------------------+------------+
```

Now, we are going to execute the below statement to add two records into the table:

1. **INSERT INTO** Student(Stud_ID, **Name**, Email, City)
2. **VALUES** (4,'Donald', 'donald@javatpoint.com', 'New York'),
3. (5, 'Joseph', 'Joseph@javatpoint.com', 'Chicago');

After the above statement's execution, we will see the error: ***ERROR 1062 (23000): Duplicate entry 'Joseph@javatpoint.com' for key 'student.Email'*** because of the email violets the UNIQUE constraint.

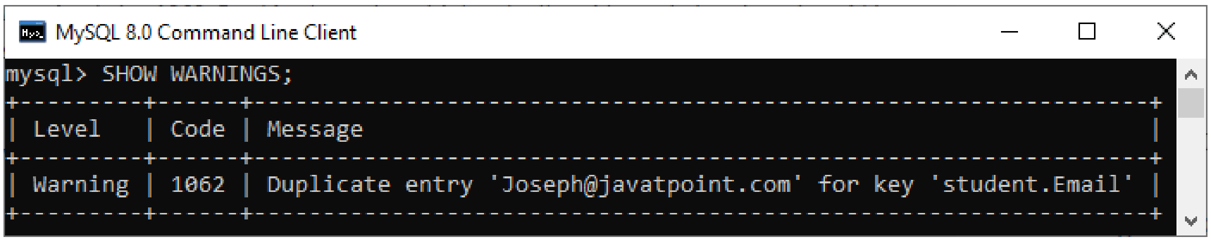
But, when we perform the same statement using the INSERT IGNORE command, we have not received any error. Instead, we receive a warning only.

1. **INSERT IGNORE INTO** Student(Stud_ID, **Name**, Email, City)
2. **VALUES** (4,'Donald', 'donald@javatpoint.com', 'New York'),
3. (5, 'Joseph', 'Joseph@javatpoint.com', 'Chicago');

MySQL will produce a message: one row added, and the other row was ignored.

1. 1 row affected, 1 warning(s): 1062 Duplicate entry **for key** email.
2. Records: 2 Duplicates: 1 Warning: 1

We can see the detailed warning using the **SHOW WARNINGS** command:



```
mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------+
| Level   | Code | Message                                                         |
+---------+------+-----------------------------------------------------------------+
| Warning | 1062 | Duplicate entry 'Joseph@javatpoint.com' for key 'student.Email' |
+---------+------+-----------------------------------------------------------------+
```

Thus, if there are some duplicates and we use the INSERT IGNORE statement, MySQL gives a warning instead of issuing an error and will add the remaining records to the table.

UPSERT using REPLACE

In some situations, we want to update the existing records into the table to keep them updated. If we use the standard insert query for this purpose, it will give a Duplicate entry for PRIMARY KEY error. In this case, we will use the REPLACE statement to perform our task. When we use the REPLACE command, there are **two possible** events take place:

- If no matching value is found with the existing data row, then a standard INSERT statement is performed.
- If the old record matches the new record, the replace command will delete the existing row, and then a normal INSERT statement is executed that adds the new record in the table.

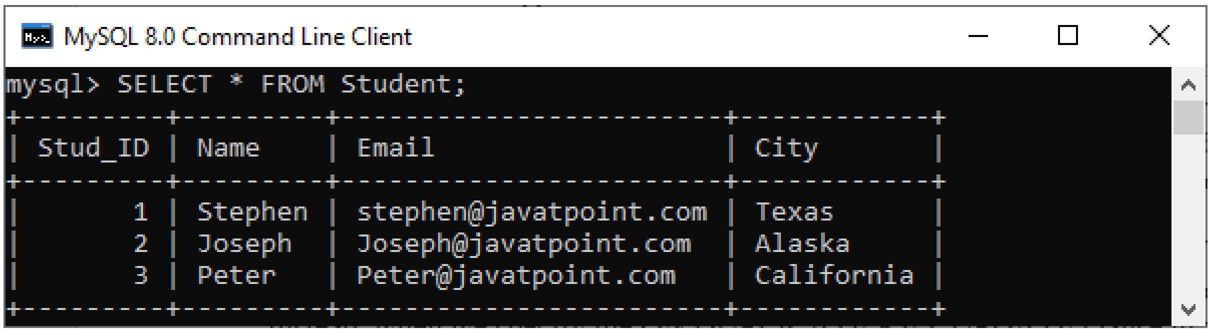
In the REPLACE statement, the updation performed in two steps. First, it will delete the existing record, and then the newly updated record is added, similar to a standard INSERT command. Thus, we can say that the REPLACE statement performs two standard functions, **DELETE** and **INSERT**.

The syntax of **REPLACE** statement in MySQL is as follows:

1. REPLACE [**INTO**] table_name(column_list)
2. **VALUES**(value_list);

Example

Let us understand it with the help of a real example. Here, we will take a **Student** table that we have created earlier that contains the following data:



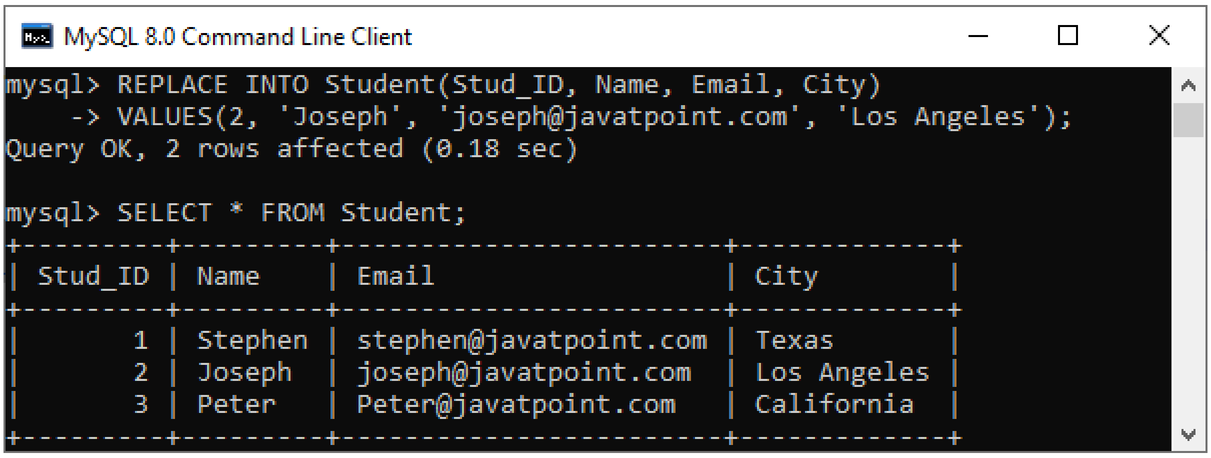
MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM Student;
+---------+---------+------------------------+------------+
| Stud_ID | Name    | Email                  | City       |
+---------+---------+------------------------+------------+
|       1 | Stephen | stephen@javatpoint.com | Texas      |
|       2 | Joseph  | Joseph@javatpoint.com  | Alaska     |
|       3 | Peter   | Peter@javatpoint.com   | California |
+---------+---------+------------------------+------------+
```

Now, we will use the REPLACE statement to update the **city of the student whose id = 2** with the new city **Los Angeles**. To do this, execute the following statement:

1. REPLACE **INTO** Student(Stud_ID, **Name**, Email, City)
2. **VALUES**(2, 'Joseph', 'joseph@javatpoint.com', 'Los Angeles');

After the successful execution, we will get the output as below:



MySQL 8.0 Command Line Client

```
mysql> REPLACE INTO Student(Stud_ID, Name, Email, City)
    -> VALUES(2, 'Joseph', 'joseph@javatpoint.com', 'Los Angeles');
Query OK, 2 rows affected (0.18 sec)

mysql> SELECT * FROM Student;
+---------+---------+------------------------+-------------+
| Stud_ID | Name    | Email                  | City        |
+---------+---------+------------------------+-------------+
|       1 | Stephen | stephen@javatpoint.com | Texas       |
|       2 | Joseph  | joseph@javatpoint.com  | Los Angeles |
|       3 | Peter   | Peter@javatpoint.com   | California  |
+---------+---------+------------------------+-------------+
```

In the above image, we can see the message that says **"2 row(s) affected"** while we have updated the values of a single row only. It is because the REPLACE command first deleted the record, and then the added the new record into the table. Hence the message says, "2 row(s) affected."

UPSERT using INSERT ON DUPLICATE KEY UPDATE

We have seen the two UPSERT commands so far, but they had some limitations. The INSERT IGNORE statement only ignores the duplicate error without making any modification to the table. And the REPLACE method detected the INSERT error, but it will delete the row before adding the new record. Hence, we are still searching for a more refined solution until now.

So, we use a more refined solution as the INSERT ON DUPLICATE KEY UPDATE statement. It is a non-destructive method that means it does not remove the duplicate row. Instead, when we specify the ON DUPLICATE KEY UPDATE clause in a SQL statement and a row would cause duplicate error value in a ***UNIQUE or PRIMARY KEY*** index column, then updation of the existing row occurs.

The syntax of **Insert on Duplicate Key Update** statement in MySQL is given below:

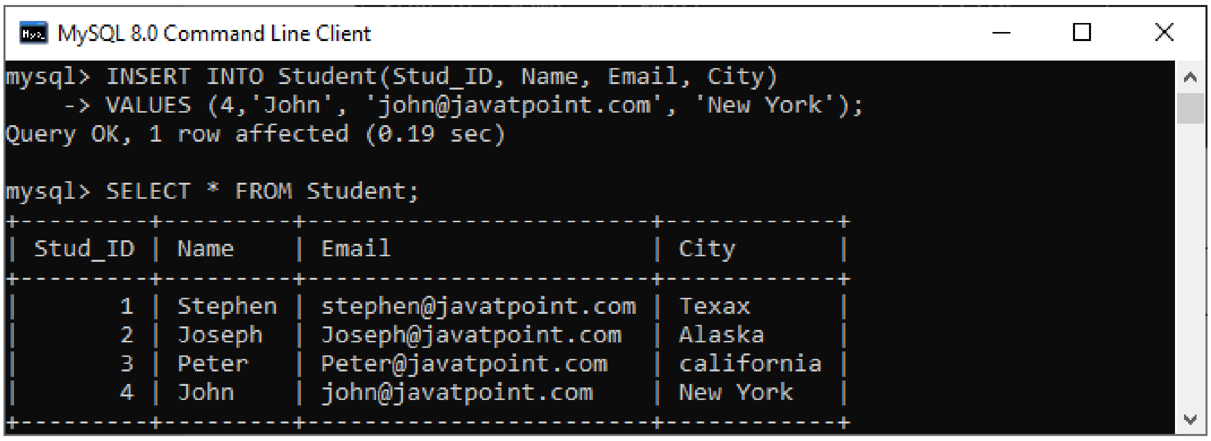
1. **INSERT INTO table** (column_names)
2. **VALUES** (data)
3. **ON** DUPLICATE **KEY UPDATE**
4. column1 = expression, column2 = expression…;

Example

Let us understand it with the help of a real example. Here, we will take a **Student** table that we have created earlier. Now, add one more row into the table using the below query:

1. **INSERT INTO** Student(Stud_ID, **Name**, Email, City)
2. **VALUES** (4,'John', 'john@javatpoint.com', 'New York');

This query will add one record successfully into the table because it does not have any duplicate values.



```
mysql> INSERT INTO Student(Stud_ID, Name, Email, City)
    -> VALUES (4,'John', 'john@javatpoint.com', 'New York');
Query OK, 1 row affected (0.19 sec)

mysql> SELECT * FROM Student;
+---------+---------+-------------------------+------------+
| Stud_ID | Name    | Email                   | City       |
+---------+---------+-------------------------+------------+
|       1 | Stephen | stephen@javatpoint.com  | Texax      |
|       2 | Joseph  | Joseph@javatpoint.com   | Alaska     |
|       3 | Peter   | Peter@javatpoint.com    | california |
|       4 | John    | john@javatpoint.com     | New York   |
+---------+---------+-------------------------+------------+
```

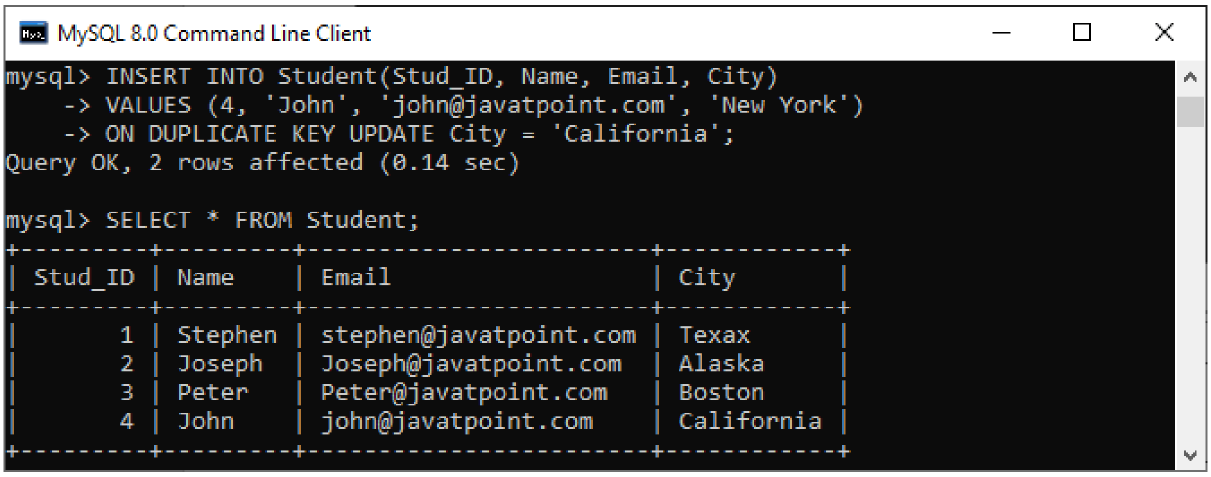
Next, execute the below MySQL UPSERT command that updated the duplicate record in the **Stud_ID** column:

1. **INSERT INTO** Student(Stud_ID, **Name**, Email, City)
2. **VALUES** (4, 'John', 'john@javatpoint.com', 'New York')
3. **ON** DUPLICATE **KEY UPDATE** City = 'California';

After successful execution, MySQL gives the following message:

1. Query OK, 2 **rows** affected.

In the output, we can see that the **row id=4** already exists. So the query only updates the **City New York** with **California**.



```
mysql> INSERT INTO Student(Stud_ID, Name, Email, City)
    -> VALUES (4, 'John', 'john@javatpoint.com', 'New York')
    -> ON DUPLICATE KEY UPDATE City = 'California';
Query OK, 2 rows affected (0.14 sec)

mysql> SELECT * FROM Student;
+---------+---------+-------------------------+------------+
| Stud_ID | Name    | Email                   | City       |
+---------+---------+-------------------------+------------+
|       1 | Stephen | stephen@javatpoint.com  | Texax      |
|       2 | Joseph  | Joseph@javatpoint.com   | Alaska     |
|       3 | Peter   | Peter@javatpoint.com    | Boston     |
|       4 | John    | john@javatpoint.com     | California |
+---------+---------+-------------------------+------------+
```