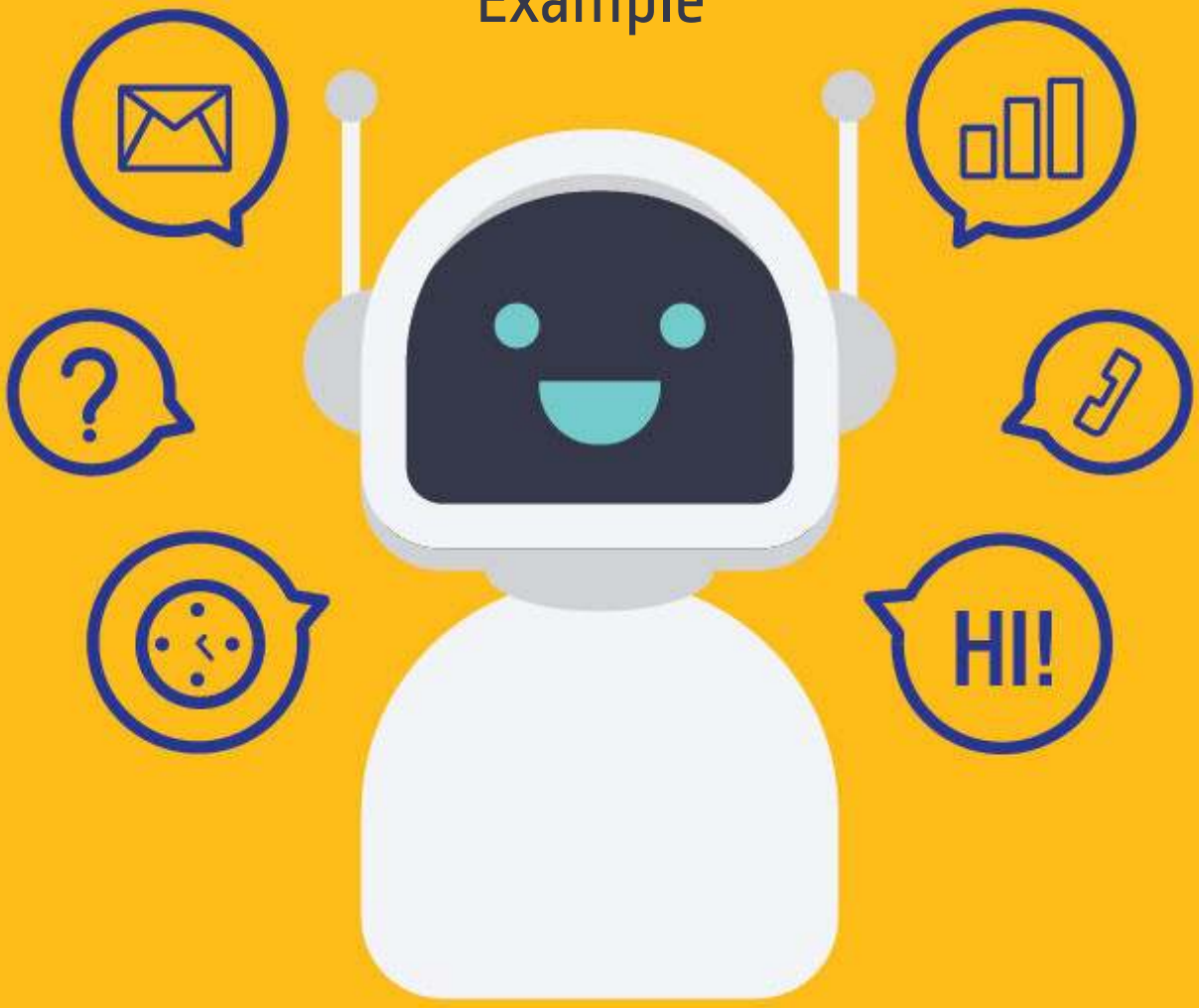
SQL for Citizen Data Scientists

SQL INSERT INTO Statement - Learn By Example



SETScholars & WACAMLDS

The INSERT INTO statement allows you to insert one or more new rows into a table. Using the INSERT INTO statement, it is possible to insert a complete row, a partial row, multiple rows or rows generated as the result of a query.

Let's now take a look at each of these in detail.

Sample Table

To help you better understand the examples, and enable you to follow along with the tutorial, we are going to use the following sample table.

This table is part of an 'Employee Management System' that contains basic information about employees.

ID	Name	Age	City	Job	Salary
1	Bob	28	New York	Manager	60000
2	Eve	24	New York	Developer	32000
3	Kim	25	Chicago	Manager	55000
4	Joe	23	Chicago	Developer	30000

INSERT a Single Row

To use INSERT INTO you must, at a minimum, specify two pieces of information – the table name and the values to be inserted into the new row. However, while doing this you must provide a value for each column and also ensure that those values are in the same order as the columns in the table definition.

Here's the basic INSERT syntax.

```
INSERT INTO table_name
VALUES (value1,value2, ..);
```

To demonstrate, let's insert a new employee into the 'Employees' table.

```
INSERT INTO Employees
VALUES(5, 'Max', 26, 'New York', 'Developer', 32000);
```

ID	Name	Age	City	Job	Salary
1	Bob	28	New York	Manager	60000
2	Eve	24	New York	Developer	32000
3	Kim	25	Chicago	Manager	55000
4	Joe	23	Chicago	Developer	30000
5	Max	26	New York	Developer	32000

Caution!

Although this syntax is simple, it is not safe and should generally be avoided; as it is highly dependent on the order in which the columns are defined in the table.

Even if you know the order now, there is no guarantee that the columns will be in the same order when the table is rebuilt next time.

The safer and recommended way to use the INSERT INTO statement is to specify the column names explicitly.

```
INSERT INTO table_name (column1,column2, ...)
VALUES (value1,value2, ...);
```

Just keep in mind that the values must be listed in the same order as the column names. This is because when the row is inserted, the DBMS will match each item in the column list with the appropriate value in the VALUES list. The first column name matches the first value. The second column name matches the second value, and so on.

The following example works exactly the same as the previous INSERT statement, but this time the column names are explicitly specified after the table name.

```
INSERT INTO Employees (ID, Name, Age, City, Job, Salary)
VALUES(5, 'Max', 26, 'New York', 'Developer', 32000);
```

ID	Name	Age	City	Job	Salary
1	Bob	28	New York	Manager	60000
2	Eve	24	New York	Developer	32000
3	Kim	25	Chicago	Manager	55000
4	Joe	23	Chicago	Developer	30000
5	Max	26	New York	Developer	32000

Because the column names are explicitly specified, you can place them in any order. The following INSERT statement is the same as before, but specifies a column list in a different order.

```
INSERT INTO Employees (City, Name, Job, Age, Salary, ID)
VALUES('New York', 'Max', 'Developer', 26, 32000, 5);
```

ID	Name	Age	City	Job	Salary
1	Bob	28	New York	Manager	60000
2	Eve	24	New York	Developer	32000
3	Kim	25	Chicago	Manager	55000
4	Joe	23	Chicago	Developer	30000
5	Max	26	New York	Developer	32000

It is a good idea to name all columns into which you are inserting values because:

- Your INSERT statement becomes more descriptive.
- You can verify that you are providing the values in the proper order based on the column names.
- You can have better data independence. The order in which the columns are defined in the table doesn't affect your INSERT statement.
- The insertion will work correctly even if the table layout changes in the future.

Inserting Partial Rows

Using the second syntax, you can omit some columns and populate the row partially. This means that you only provide values for some columns, but not for others. The DBMS will insert a NULL or a default value into any column for which you do not specify a value.

To demonstrate let's insert a new row omitting the 'Age' and 'City' columns and their corresponding values.

```
INSERT INTO Employees (ID, Name, Job, Salary)
VALUES(5, 'Max', 'Janitor', 9000);
```

ID	Name	Age	City	Job	Salary
1	Bob	28	New York	Manager	60000
2	Eve	24	New York	Developer	32000
3	Kim	25	Chicago	Manager	55000
4	Joe	23	Chicago	Developer	30000
5	Max	*NULL*	*NULL*	Janitor	9000

Waring!

If you plan to skip a column, then you need to make sure that one of the following conditions exists for that column:

- The column is defined to allow NULL values.
- A default value for the column is specified in the table definition.

If you omit a column that does not allow NULL values and does not have a default value, then the INSERT statement fails and an error is generated.

INSERT Multiple Rows at once

If you have thousands of records to insert, you don't need to execute one INSERT at a time. You can specify multiple records in a single INSERT statement. Simply list the values of each row inside the pair of parentheses and separate each list from the others using a comma:

```
INSERT INTO table_name (column1,column2, ..)
VALUES (value1,value2, ..),
    (value1,value2, ..),
    ..
    (value1,value2, ..);
```

The following INSERT statement inserts two rows at the same time in the 'Employees' table.

```
INSERT INTO Employees (ID, Name, Age, City, Job, Salary)
VALUES(5, 'Max', 26, 'New York', 'Janitor', 9000),
      (6, 'Sam', 27, 'Chicago', 'Janitor', 10000);
```

ID	Name	Age	City	Job	Salary
1	Bob	28	New York	Manager	60000
2	Eve	24	New York	Developer	32000
3	Kim	25	Chicago	Manager	55000
4	Joe	23	Chicago	Developer	30000
5	Max	26	New York	Janitor	9000
6	Sam	27	Chicago	Janitor	10000

It is more efficient to do multiple inserts in this way, especially if you have thousands of records. Otherwise, the insertion process can run very slowly.