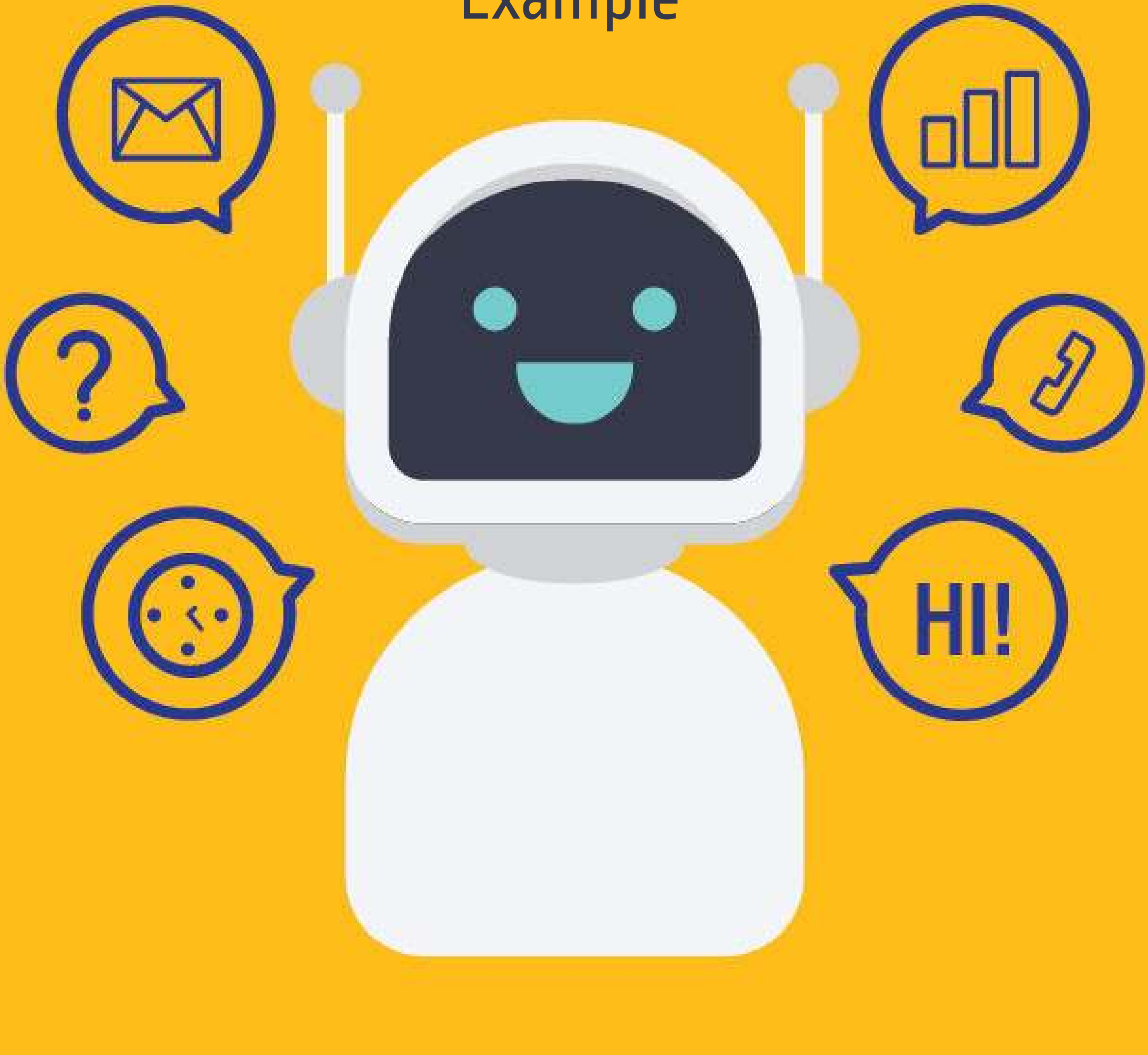
SQL for Citizen Data Scientists

SQL SELECT INTO Statement - Learn By Example



SETScholars & WACAMLDS

There is another way to insert data into a table that does not use INSERT statement at all – The SELECT INTO statement. This statement is used to copy the contents of a table to a new table.

Unlike INSERT SELECT, which adds data to an existing table, SELECT INTO copies the data into a brand new table, created on the fly.

SELECT INTO is a great way to create a new identical copy of a database, for example, for backup purposes or to test your queries on a copy instead of live data or simply you want to move the database to a new computer.

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	Max	26	New York	Janitor	9000
4	Kim	25	Chicago	Manager	55000
5	Joe	23	Chicago	Developer	30000
6	Sam	27	Chicago	Janitor	10000

Copy Entire Table

SELECT INTO has two different parts. First, it creates a new table with columns corresponding to the columns listed in the SELECT list. Second, it copies the entire contents of the source table into the new table.

The name of the new table appears with the INTO clause, and the name of the source table appears in the FROM clause of the SELECT statement. Here's the basic SELECT INTO syntax.

```
SELECT *
INTO new_table
FROM source_table;
```

This following SELECT INTO statement creates a new table named 'Employees_Backup' and copies the entire contents of the 'Employees' table into it:

```
SELECT *
INTO Employees_Backup
FROM Employees;
```

After running this query, the contents of the 'Employees_Backup' table will be:

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

Copy Selected Rows

The SELECT statement embedded in the SELECT INTO statement is no different from the SELECT statement you use to retrieve data, so it can include a WHERE clause. And this WHERE clause filters the data to be copied.

```
SELECT *
INTO new_table
FROM source_table
WHERE condition;
```

The following SELECT INTO statement only copies 'New York' employees to a new table.

```
SELECT *
INTO NYEmployees
FROM Employees
WHERE City = 'New York';
```

After running this query, the contents of the 'NYEmployees' table will be:

ID	Name	Age	City	Job	Salary
1	Bob	28	New York	Manager	60000
2	Eve	24	New York	Developer	32000
3	Max	26	New York	Janitor	9000

Copy Selected Columns

Since SELECT * was used in the above examples, each column in the 'Employees' table was copied to the new table. To copy only a subset of columns, column names must be explicitly specified.

```
SELECT column1,column2,column3, ..
INTO new_table
FROM source_table;
```

The following SELECT INTO statement copies only three columns into a new table:

```
SELECT ID, Name, Job
INTO Employees_Backup
FROM Employees;
```

After running this query, the contents of the 'Employees_Backup' table will be:

ID	Name	Job
1	Bob	Manager
2	Eve	Developer
3	Max	Janitor
4	Kim	Manager
5	Joe	Developer
6	Sam	Janitor

Assigning Aliases to Columns in New Tables

As you can see the new tables were created with column names and types as defined in the ‘Employees’ table. However, you can create a new table with new column names using column aliases (the AS clause).

Here is an extension of the previous query that has been modified to use aliases:

```
SELECT ID AS 'New_ID', Name AS 'New_Name', Job AS 'New_Job'
INTO Employees_Backup
FROM Employees;
```

After running this query, the contents of the ‘Employees_Backup’ table will be:

New_ID	New_Name	New_Job
1	Bob	Manager
2	Eve	Developer
3	Max	Janitor
4	Kim	Manager
5	Joe	Developer
6	Sam	Janitor

Create an Empty Table

SELECT INTO can also be used to create an empty table using the schema of another. Just add a WHERE clause that causes the query to return nothing:

```
SELECT *
INTO Employees_Backup
FROM Employees
WHERE 1 = 0;
```

After running this query, the contents of the 'Employees_Backup' table will be: