**WEEK 4**

**Task 1: SQL for Data Analytics**

SQL is a **key cog** in a data science professional’s armory. We are speaking from our experience. You simply cannot expect to carve out a successful career in either analytics or data science if you haven’t yet picked up SQL.

Most people tend to dive right into complex technologies like ML & AI, but majority of people working in top tech companies stated that **SQL accounts for 85% of day to day tasks of a Data Analyst**. Not only this, every company puts **heavy emphasis on SQL during the interviews** and assignments before hiring a candidate.

Therefore we will be covering basics of SQL using the course linked below.

**SQL Resource**:

If you are a complete beginner with no experience in SQL, you may find the resource below useful. Rest of you can move forward to the next module, so we can focus right into SQL practice along with the **most widely used queries and functions in companies** and **often asked in interviews.**

Video - <https://youtu.be/7S_tz1z_5bA>

**Task 2: Basic Querying in SQL**

Now that you have complete understanding of SQL from the previous module, let’s **get to practice.** In this module, we are going to revisit the **most fundamental and important SQL commands** and practice along the way. Start completing, each sub module to get a strong grasp over SQL for this week’s upcoming mini project on SQL

# Select

## The SQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

## SELECT Syntax

SELECT column1, column2, ...

FROM table\_name;

Here, column1, column2, … are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT \* FROM table\_name;

## Demo Data Base

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SELECT Column Example

The following SQL statement selects the “CustomerName” and “City” columns from the “Customers” table:

### Example

SELECT CustomerName, City From Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_columns)

## SELECT \* Example

The following SQL statement selects all the columns from the “Customers” table:

### Example

SELECT \* FROM Customers;

# Select Distinct

## The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

### SELECT DISTINCT Syntax

SELECT DISTINCT column1, column2, ...

FROM table\_name;

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SELECT Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the “Country” column in the “Customers” table:

### Example

SELECT Country FROM Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_no_distinct)

Now, let us use the SELECT DISTINCT statement and see the result.

## SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the “Country” column in the “Customers” table:

### Example

SELECT DISTINCT Country FROM Customers;

[Try it Yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_distinct)

The following SQL statement lists the number of different (distinct) customer countries:

### Example

SELECT COUNT(DISTINCT Country) FROM Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_distinct2)

# Where clause

## The SQL WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

### WHERE Syntax

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

**Note:** The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## WHERE Clause Example

The following SQL statement selects all the customers from the country “Mexico”, in the “Customers” table:

## Example

SELECT \* FROM Customers

WHERE Country='Mexico';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where)

## Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

### Example

SELECT \* FROM Customers

WHERE CustomerID=1;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_number)

## Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_equal_to) |
| > | Greater than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than) |
| < | Less than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than) |
| >= | Greater than or equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than2) |
| <= | Less than or equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than2) |
| <> | Not equal. **Note:** In some versions of SQL this operator may be written as != | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not_equal_to) |
| BETWEEN | Between a certain range | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_between) |
| LIKE | Search for a pattern | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_like) |
| IN | To specify multiple possible values for a column | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in) |

# AND , OR , NOT

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [AND , OR , NOT](https://dashboard.stige.in/index.php/topic/and-or-not/)

In Progress

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

* The AND operator displays a record if all the conditions separated by AND are TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

### AND Syntax

SELECT column1, column2, ...

FROM table\_name

WHERE condition1 AND condition2 AND condition3 ...;

### OR Syntax

SELECT column1, column2, ...

FROM table\_name

WHERE condition1 OR condition2 OR condition3 ...;

### NOT Syntax

SELECT column1, column2, ...

FROM table\_name

WHERE NOT condition;

## Demo Database

The table below shows the complete “Customers” table from the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app’ | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |
| 11 | B’s Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 13 | Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. | 05022 | Mexico |
| 14 | Chop-suey Chinese | Yang Wang | Hauptstr. 29 | Bern | 3012 | Switzerland |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |
| 18 | Du monde entier | Janine Labrune | 67, rue des Cinquante Otages | Nantes | 44000 | France |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |
| 22 | FISSA Fabrica Inter. Salchichas S.A. | Diego Roel | C/ Moralzarzal, 86 | Madrid | 28034 | Spain |
| 23 | Folies gourmandes | Martine Rancé | 184, chaussée de Tournai | Lille | 59000 | France |
| 24 | Folk och fä HB | Maria Larsson | Åkergatan 24 | Bräcke | S-844 67 | Sweden |
| 25 | Frankenversand | Peter Franken | Berliner Platz 43 | München | 80805 | Germany |
| 26 | France restauration | Carine Schmitt | 54, rue Royale | Nantes | 44000 | France |
| 27 | Franchi S.p.A. | Paolo Accorti | Via Monte Bianco 34 | Torino | 10100 | Italy |
| 28 | Furia Bacalhau e Frutos do Mar | Lino Rodriguez | Jardim das rosas n. 32 | Lisboa | 1675 | Portugal |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 31 | Gourmet Lanchonetes | André Fonseca | Av. Brasil, 442 | Campinas | 04876-786 | Brazil |
| 32 | Great Lakes Food Market | Howard Snyder | 2732 Baker Blvd. | Eugene | 97403 | USA |
| 33 | GROSELLA-Restaurante | Manuel Pereira | 5ª Ave. Los Palos Grandes | Caracas | 1081 | Venezuela |
| 34 | Hanari Carnes | Mario Pontes | Rua do Paço, 67 | Rio de Janeiro | 05454-876 | Brazil |
| 35 | HILARIÓN-Abastos | Carlos Hernández | Carrera 22 con Ave. Carlos Soublette #8-35 | San Cristóbal | 5022 | Venezuela |
| 36 | Hungry Coyote Import Store | Yoshi Latimer | City Center Plaza 516 Main St. | Elgin | 97827 | USA |
| 37 | Hungry Owl All-Night Grocers | Patricia McKenna | 8 Johnstown Road | Cork |  | Ireland |
| 38 | Island Trading | Helen Bennett | Garden House Crowther Way | Cowes | PO31 7PJ | UK |
| 39 | Königlich Essen | Philip Cramer | Maubelstr. 90 | Brandenburg | 14776 | Germany |
| 40 | La corne d’abondance | Daniel Tonini | 67, avenue de l’Europe | Versailles | 78000 | France |
| 41 | La maison d’Asie | Annette Roulet | 1 rue Alsace-Lorraine | Toulouse | 31000 | France |
| 42 | Laughing Bacchus Wine Cellars | Yoshi Tannamuri | 1900 Oak St. | Vancouver | V3F 2K1 | Canada |
| 43 | Lazy K Kountry Store | John Steel | 12 Orchestra Terrace | Walla Walla | 99362 | USA |
| 44 | Lehmanns Marktstand | Renate Messner | Magazinweg 7 | Frankfurt a.M. | 60528 | Germany |
| 45 | Let’s Stop N Shop | Jaime Yorres | 87 Polk St. Suite 5 | San Francisco | 94117 | USA |
| 46 | LILA-Supermercado | Carlos González | Carrera 52 con Ave. Bolívar #65-98 Llano Largo | Barquisimeto | 3508 | Venezuela |
| 47 | LINO-Delicateses | Felipe Izquierdo | Ave. 5 de Mayo Porlamar | I. de Margarita | 4980 | Venezuela |
| 48 | Lonesome Pine Restaurant | Fran Wilson | 89 Chiaroscuro Rd. | Portland | 97219 | USA |
| 49 | Magazzini Alimentari Riuniti | Giovanni Rovelli | Via Ludovico il Moro 22 | Bergamo | 24100 | Italy |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium |
| 51 | Mère Paillarde | Jean Fresnière | 43 rue St. Laurent | Montréal | H1J 1C3 | Canada |
| 52 | Morgenstern Gesundkost | Alexander Feuer | Heerstr. 22 | Leipzig | 04179 | Germany |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina |
| 55 | Old World Delicatessen | Rene Phillips | 2743 Bering St. | Anchorage | 99508 | USA |
| 56 | Ottilies Käseladen | Henriette Pfalzheim | Mehrheimerstr. 369 | Köln | 50739 | Germany |
| 57 | Paris spécialités | Marie Bertrand | 265, boulevard Charonne | Paris | 75012 | France |
| 58 | Pericles Comidas clásicas | Guillermo Fernández | Calle Dr. Jorge Cash 321 | México D.F. | 05033 | Mexico |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 60 | Princesa Isabel Vinhoss | Isabel de Castro | Estrada da saúde n. 58 | Lisboa | 1756 | Portugal |
| 61 | Que Delícia | Bernardo Batista | Rua da Panificadora, 12 | Rio de Janeiro | 02389-673 | Brazil |
| 62 | Queen Cozinha | Lúcia Carvalho | Alameda dos Canàrios, 891 | São Paulo | 05487-020 | Brazil |
| 63 | QUICK-Stop | Horst Kloss | Taucherstraße 10 | Cunewalde | 01307 | Germany |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 65 | Rattlesnake Canyon Grocery | Paula Wilson | 2817 Milton Dr. | Albuquerque | 87110 | USA |
| 66 | Reggiani Caseifici | Maurizio Moroni | Strada Provinciale 124 | Reggio Emilia | 42100 | Italy |
| 67 | Ricardo Adocicados | Janete Limeira | Av. Copacabana, 267 | Rio de Janeiro | 02389-890 | Brazil |
| 68 | Richter Supermarkt | Michael Holz | Grenzacherweg 237 | Genève | 1203 | Switzerland |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |
| 70 | Santé Gourmet | Jonas Bergulfsen | Erling Skakkes gate 78 | Stavern | 4110 | Norway |
| 71 | Save-a-lot Markets | Jose Pavarotti | 187 Suffolk Ln. | Boise | 83720 | USA |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |
| 73 | Simons bistro | Jytte Petersen | Vinbæltet 34 | København | 1734 | Denmark |
| 74 | Spécialités du monde | Dominique Perrier | 25, rue Lauriston | Paris | 75016 | France |
| 75 | Split Rail Beer & Ale | Art Braunschweiger | P.O. Box 555 | Lander | 82520 | USA |
| 76 | Suprêmes délices | Pascale Cartrain | Boulevard Tirou, 255 | Charleroi | B-6000 | Belgium |
| 77 | The Big Cheese | Liz Nixon | 89 Jefferson Way Suite 2 | Portland | 97201 | USA |
| 78 | The Cracker Box | Liu Wong | 55 Grizzly Peak Rd. | Butte | 59801 | USA |
| 79 | Toms Spezialitäten | Karin Josephs | Luisenstr. 48 | Münster | 44087 | Germany |
| 80 | Tortuga Restaurante | Miguel Angel Paolino | Avda. Azteca 123 | México D.F. | 05033 | Mexico |
| 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil |
| 82 | Trail’s Head Gourmet Provisioners | Helvetius Nagy | 722 DaVinci Blvd. | Kirkland | 98034 | USA |
| 83 | Vaffeljernet | Palle Ibsen | Smagsløget 45 | Århus | 8200 | Denmark |
| 84 | Victuailles en stock | Mary Saveley | 2, rue du Commerce | Lyon | 69004 | France |
| 85 | Vins et alcools Chevalier | Paul Henriot | 59 rue de l’Abbaye | Reims | 51100 | France |
| 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |

## AND Example

The following SQL statement selects all fields from “Customers” where country is “Germany” AND city is “Berlin”:

### Example

SELECT \* FROM Customers

WHERE Country='Germany' AND City='Berlin';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_and)

## OR Example

The following SQL statement selects all fields from “Customers” where city is “Berlin” OR “München”:

### Example

SELECT \* FROM Customers

WHERE City='Berlin' OR City='München';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_or)

The following SQL statement selects all fields from “Customers” where country is “Germany” OR “Spain”:

### Example

SELECT \* FROM Customers

WHERE Country='Germany' OR Country='Spain';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_or2)

## NOT Example

The following SQL statement selects all fields from “Customers” where country is NOT “Germany”:

### Example

SELECT \* FROM Customers

WHERE NOT Country='Germany';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_not)

## Combining AND, OR and NOT

You can also combine the AND, OR and NOT operators.

The following SQL statement selects all fields from “Customers” where country is “Germany” AND city must be “Berlin” OR “München” (use parenthesis to form complex expressions):

### Example

SELECT \* FROM Customers

WHERE Country='Germany' AND (City='Berlin' OR City='München');

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_and_or)

The following SQL statement selects all fields from “Customers” where country is NOT “Germany” and NOT “USA”:

### Example

SELECT \* FROM Customers

WHERE NOT Country='Germany' AND NOT Country='USA';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_not_and)

# Insert Intro

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Insert Intro](https://dashboard.stige.in/index.php/topic/insert-intro/)

In Progress

The INSERT INTO statement is used to insert new records in a table.

### INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO table\_name

VALUES (value1, value2, value3, ...);

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 89 | White Clover Markets | Karl Jablonski | 305 – 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

## INSERT INTO Example

The following SQL statement inserts a new record in the “Customers” table:

### Example

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_insert_colname)

The selection from the “Customers” table will now look like this:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 89 | White Clover Markets | Karl Jablonski | 305 – 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 92 | Cardinal | Tom B. Erichsen | Skagen 21 | Stavanger | 4006 | Norway |

**Did you notice that we did not insert any number into the CustomerID field?**  
The CustomerID column is an [auto-increment](https://www.w3schools.com/sql/sql_autoincrement.asp) field and will be generated automatically when a new record is inserted into the table.

## Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the “CustomerName”, “City”, and “Country” columns (CustomerID will be updated automatically):

### Example

INSERT INTO Customers (CustomerName, City, Country)

VALUES ('Cardinal', 'Stavanger', 'Norway');

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_insert_cols)

The selection from the “Customers” table will now look like this:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 89 | White Clover Markets | Karl Jablonski | 305 – 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 92 | Cardinal | null | null | Stavanger | null | Norway |

# NULL Value

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [NULL Value](https://dashboard.stige.in/index.php/topic/null-value/)

In Progress

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

**Note:** A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

## How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

### IS NULL Syntax

SELECT column\_names

FROM table\_name

WHERE column\_name IS NULL;

### IS NOT NULL Syntax

SELECT column\_names

FROM table\_name

WHERE column\_name IS NOT NULL;

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Swe |

## The IS NULL Operator

The IS NULL operator is used to test for empty values (NULL values).

The following SQL lists all customers with a NULL value in the “Address” field:

### Example

SELECT CustomerName, ContactName, Address

FROM Customers

WHERE Address IS NULL;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_is_null)

**Tip:** Always use IS NULL to look for NULL values.

## The IS NOT NULL Operator

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).

The following SQL lists all customers with a value in the “Address” field:

### Example

SELECT CustomerName, ContactName, Address

FROM Customers

WHERE Address IS NOT NULL;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_is_not_null)

# Update

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Update](https://dashboard.stige.in/index.php/topic/update/)

In Progress

The UPDATE statement is used to modify the existing records in a table.

### UPDATE Syntax

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

**Note:** Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person and a new city.

### Example

UPDATE Customers

SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'

WHERE CustomerID = 1;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_update_2)

The selection from the “Customers” table will now look like this:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Alfred Schmidt | Obere Str. 57 | Frankfurt | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the ContactName to “Juan” for all records where country is “Mexico”:

### Example

UPDATE Customers

SET ContactName='Juan'

WHERE Country='Mexico';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_update_3)

The selection from the “Customers” table will now look like this:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Alfred Schmidt | Obere Str. 57 | Frankfurt | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Juan | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Juan | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## Update Warning!

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

### Example

UPDATE Customers

SET ContactName='Juan';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_update_4)

The selection from the “Customers” table will now look like this:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Juan | Obere Str. 57 | Frankfurt | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Juan | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Juan | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Juan | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Juan | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

# Delete

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Delete](https://dashboard.stige.in/index.php/topic/delete/)

In Progress

The DELETE statement is used to delete existing records in a table.

DELETE FROM table\_name WHERE condition;

**Note:** Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL DELETE Example

The following SQL statement deletes the customer “Alfreds Futterkiste” from the “Customers” table:

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_delete)

The “Customers” table will now look like this:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM table\_name;

The following SQL statement deletes all rows in the “Customers” table, without deleting the table:

DELETE FROM Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_delete_all)

# Select Top

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Select Top](https://dashboard.stige.in/index.php/topic/select-top/)

In Progress

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

**Note:** Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.

**SQL Server / MS Access Syntax:**

SELECT TOP number|percent column\_name(s)

FROM table\_name

WHERE condition;

**MySQL Syntax:**

SELECT column\_name(s)

FROM table\_name

WHERE condition

LIMIT number;

**Oracle 12 Syntax:**

SELECT column\_name(s)

FROM table\_name

ORDER BY column\_name(s)

FETCH FIRST number ROWS ONLY;

**Older Oracle Syntax:**

SELECT column\_name(s)

FROM table\_name

WHERE ROWNUM <= number;

**Older Oracle Syntax (with ORDER BY):**

SELECT \*

FROM (SELECT column\_name(s) FROM table\_name ORDER BY column\_name(s))

WHERE ROWNUM <= number;

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL TOP, LIMIT and FETCH FIRST Examples

The following SQL statement selects the first three records from the “Customers” table (for SQL Server/MS Access):

### Example

SELECT TOP 3 \* FROM Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_top&ss=-1)

The following SQL statement shows the equivalent example for MySQL:

### Example

**SELECT \* FROM** Customers

LIMIT 3;

[Try it yourself](https://www.w3schools.com/sql/trymysql.asp?filename=trysql_select_limit)

The following SQL statement shows the equivalent example for Oracle:

### Example

SELECT \* FROM Customers

FETCH FIRST 3 ROWS ONLY;

## SQL TOP PERCENT Example

The following SQL statement selects the first 50% of the records from the “Customers” table (for SQL Server/MS Access):

### Example

SELECT TOP 50 PERCENT \* FROM Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_top_percent&ss=-1)

The following SQL statement shows the equivalent example for Oracle:

### Example

SELECT \* FROM Customers

FETCH FIRST 50 PERCENT ROWS ONLY;

## ADD a WHERE CLAUSE

The following SQL statement selects the first three records from the “Customers” table, where the country is “Germany” (for SQL Server/MS Access):

### Example

SELECT TOP 3 \* FROM Customers

WHERE Country='Germany';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_top_where&ss=-1)

The following SQL statement shows the equivalent example for MySQL:

### Example

SELECT TOP 3 \* FROM Customers

WHERE Country='Germany';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_top_where&ss=-1)

The following SQL statement shows the equivalent example for MySQL:

### Example

SELECT \* FROM Customers

WHERE Country='Germany'

LIMIT 3;

[Try it yourself](https://www.w3schools.com/sql/trymysql.asp?filename=trysql_select_limit_where)

The following SQL statement shows the equivalent example for Oracle:

### Example

SELECT \* FROM Customers

WHERE Country='Germany'

FETCH FIRST 3 ROWS ONLY;

# Like

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Like](https://dashboard.stige.in/index.php/topic/like/)

In Progress

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

* The percent sign (%) represents zero, one, or multiple characters
* The underscore sign (\_) represents one, single character

**Note:** MS Access uses an asterisk (\*) instead of the percent sign (%), and a question mark (?) instead of the underscore (\_).

The percent sign and the underscore can also be used in combinations!

### LIKE Syntax

SELECT column1, column2, ...

FROM table\_name

WHERE columnN LIKE pattern;

**Tip:** You can also combine any number of conditions using AND or OR operators.

Here are some examples showing different LIKE operators with ‘%’ and ‘\_’ wildcards:

| **LIKE Operator** | **Description** |
| --- | --- |
| WHERE CustomerName LIKE ‘a%’ | Finds any values that start with “a” |
| WHERE CustomerName LIKE ‘%a’ | Finds any values that end with “a” |
| WHERE CustomerName LIKE ‘%or%’ | Finds any values that have “or” in any position |
| WHERE CustomerName LIKE ‘\_r%’ | Finds any values that have “r” in the second position |
| WHERE CustomerName LIKE ‘a\_%’ | Finds any values that start with “a” and are at least 2 characters in length |
| WHERE CustomerName LIKE ‘a\_\_%’ | Finds any values that start with “a” and are at least 3 characters in length |
| WHERE ContactName LIKE ‘a%o’ | Finds any values that start with “a” and ends with “o” |

## Demo Database

The table below shows the complete “Customers” table from the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app’ | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |
| 11 | B’s Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 13 | Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. | 05022 | Mexico |
| 14 | Chop-suey Chinese | Yang Wang | Hauptstr. 29 | Bern | 3012 | Switzerland |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |
| 18 | Du monde entier | Janine Labrune | 67, rue des Cinquante Otages | Nantes | 44000 | France |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |
| 22 | FISSA Fabrica Inter. Salchichas S.A. | Diego Roel | C/ Moralzarzal, 86 | Madrid | 28034 | Spain |
| 23 | Folies gourmandes | Martine Rancé | 184, chaussée de Tournai | Lille | 59000 | France |
| 24 | Folk och fä HB | Maria Larsson | Åkergatan 24 | Bräcke | S-844 67 | Sweden |
| 25 | Frankenversand | Peter Franken | Berliner Platz 43 | München | 80805 | Germany |
| 26 | France restauration | Carine Schmitt | 54, rue Royale | Nantes | 44000 | France |
| 27 | Franchi S.p.A. | Paolo Accorti | Via Monte Bianco 34 | Torino | 10100 | Italy |
| 28 | Furia Bacalhau e Frutos do Mar | Lino Rodriguez | Jardim das rosas n. 32 | Lisboa | 1675 | Portugal |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 31 | Gourmet Lanchonetes | André Fonseca | Av. Brasil, 442 | Campinas | 04876-786 | Brazil |
| 32 | Great Lakes Food Market | Howard Snyder | 2732 Baker Blvd. | Eugene | 97403 | USA |
| 33 | GROSELLA-Restaurante | Manuel Pereira | 5ª Ave. Los Palos Grandes | Caracas | 1081 | Venezuela |
| 34 | Hanari Carnes | Mario Pontes | Rua do Paço, 67 | Rio de Janeiro | 05454-876 | Brazil |
| 35 | HILARIÓN-Abastos | Carlos Hernández | Carrera 22 con Ave. Carlos Soublette #8-35 | San Cristóbal | 5022 | Venezuela |
| 36 | Hungry Coyote Import Store | Yoshi Latimer | City Center Plaza 516 Main St. | Elgin | 97827 | USA |
| 37 | Hungry Owl All-Night Grocers | Patricia McKenna | 8 Johnstown Road | Cork |  | Ireland |
| 38 | Island Trading | Helen Bennett | Garden House Crowther Way | Cowes | PO31 7PJ | UK |
| 39 | Königlich Essen | Philip Cramer | Maubelstr. 90 | Brandenburg | 14776 | Germany |
| 40 | La corne d’abondance | Daniel Tonini | 67, avenue de l’Europe | Versailles | 78000 | France |
| 41 | La maison d’Asie | Annette Roulet | 1 rue Alsace-Lorraine | Toulouse | 31000 | France |
| 42 | Laughing Bacchus Wine Cellars | Yoshi Tannamuri | 1900 Oak St. | Vancouver | V3F 2K1 | Canada |
| 43 | Lazy K Kountry Store | John Steel | 12 Orchestra Terrace | Walla Walla | 99362 | USA |
| 44 | Lehmanns Marktstand | Renate Messner | Magazinweg 7 | Frankfurt a.M. | 60528 | Germany |
| 45 | Let’s Stop N Shop | Jaime Yorres | 87 Polk St. Suite 5 | San Francisco | 94117 | USA |
| 46 | LILA-Supermercado | Carlos González | Carrera 52 con Ave. Bolívar #65-98 Llano Largo | Barquisimeto | 3508 | Venezuela |
| 47 | LINO-Delicateses | Felipe Izquierdo | Ave. 5 de Mayo Porlamar | I. de Margarita | 4980 | Venezuela |
| 48 | Lonesome Pine Restaurant | Fran Wilson | 89 Chiaroscuro Rd. | Portland | 97219 | USA |
| 49 | Magazzini Alimentari Riuniti | Giovanni Rovelli | Via Ludovico il Moro 22 | Bergamo | 24100 | Italy |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium |
| 51 | Mère Paillarde | Jean Fresnière | 43 rue St. Laurent | Montréal | H1J 1C3 | Canada |
| 52 | Morgenstern Gesundkost | Alexander Feuer | Heerstr. 22 | Leipzig | 04179 | Germany |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina |
| 55 | Old World Delicatessen | Rene Phillips | 2743 Bering St. | Anchorage | 99508 | USA |
| 56 | Ottilies Käseladen | Henriette Pfalzheim | Mehrheimerstr. 369 | Köln | 50739 | Germany |
| 57 | Paris spécialités | Marie Bertrand | 265, boulevard Charonne | Paris | 75012 | France |
| 58 | Pericles Comidas clásicas | Guillermo Fernández | Calle Dr. Jorge Cash 321 | México D.F. | 05033 | Mexico |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 60 | Princesa Isabel Vinhoss | Isabel de Castro | Estrada da saúde n. 58 | Lisboa | 1756 | Portugal |
| 61 | Que Delícia | Bernardo Batista | Rua da Panificadora, 12 | Rio de Janeiro | 02389-673 | Brazil |
| 62 | Queen Cozinha | Lúcia Carvalho | Alameda dos Canàrios, 891 | São Paulo | 05487-020 | Brazil |
| 63 | QUICK-Stop | Horst Kloss | Taucherstraße 10 | Cunewalde | 01307 | Germany |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 65 | Rattlesnake Canyon Grocery | Paula Wilson | 2817 Milton Dr. | Albuquerque | 87110 | USA |
| 66 | Reggiani Caseifici | Maurizio Moroni | Strada Provinciale 124 | Reggio Emilia | 42100 | Italy |
| 67 | Ricardo Adocicados | Janete Limeira | Av. Copacabana, 267 | Rio de Janeiro | 02389-890 | Brazil |
| 68 | Richter Supermarkt | Michael Holz | Grenzacherweg 237 | Genève | 1203 | Switzerland |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |
| 70 | Santé Gourmet | Jonas Bergulfsen | Erling Skakkes gate 78 | Stavern | 4110 | Norway |
| 71 | Save-a-lot Markets | Jose Pavarotti | 187 Suffolk Ln. | Boise | 83720 | USA |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |
| 73 | Simons bistro | Jytte Petersen | Vinbæltet 34 | København | 1734 | Denmark |
| 74 | Spécialités du monde | Dominique Perrier | 25, rue Lauriston | Paris | 75016 | France |
| 75 | Split Rail Beer & Ale | Art Braunschweiger | P.O. Box 555 | Lander | 82520 | USA |
| 76 | Suprêmes délices | Pascale Cartrain | Boulevard Tirou, 255 | Charleroi | B-6000 | Belgium |
| 77 | The Big Cheese | Liz Nixon | 89 Jefferson Way Suite 2 | Portland | 97201 | USA |
| 78 | The Cracker Box | Liu Wong | 55 Grizzly Peak Rd. | Butte | 59801 | USA |
| 79 | Toms Spezialitäten | Karin Josephs | Luisenstr. 48 | Münster | 44087 | Germany |
| 80 | Tortuga Restaurante | Miguel Angel Paolino | Avda. Azteca 123 | México D.F. | 05033 | Mexico |
| 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil |
| 82 | Trail’s Head Gourmet Provisioners | Helvetius Nagy | 722 DaVinci Blvd. | Kirkland | 98034 | USA |
| 83 | Vaffeljernet | Palle Ibsen | Smagsløget 45 | Århus | 8200 | Denmark |
| 84 | Victuailles en stock | Mary Saveley | 2, rue du Commerce | Lyon | 69004 | France |
| 85 | Vins et alcools Chevalier | Paul Henriot | 59 rue de l’Abbaye | Reims | 51100 | France |
| 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |
| 89 | White Clover Markets | Karl Jablonski | 305 – 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

## SQL LIKE Examples

The following SQL statement selects all customers with a CustomerName starting with “a”:

### Example

SELECT \* FROM Customers

WHERE CustomerName LIKE 'a%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like)

The following SQL statement selects all customers with a CustomerName ending with “a”:

### Example

SELECT \* FROM Customers

WHERE CustomerName LIKE '%a';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_ending)

The following SQL statement selects all customers with a CustomerName that have “or” in any position:

### Example

SELECT \* FROM Customers

WHERE CustomerName LIKE '%or%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_pattern)

The following SQL statement selects all customers with a CustomerName that have “r” in the second position:

### Example

SELECT \* FROM Customers

WHERE CustomerName LIKE '\_r%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_underscore)

The following SQL statement selects all customers with a CustomerName that starts with “a” and are at least 3 characters in length:

### Example

SELECT \* FROM Customers

WHERE CustomerName LIKE 'a\_\_%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_start_least)

The following SQL statement selects all customers with a ContactName that starts with “a” and ends with “o”:

### Example

SELECT \* FROM Customers

WHERE ContactName LIKE 'a%o';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_start_end)

The following SQL statement selects all customers with a CustomerName that does NOT start with “a”:

### Example

SELECT \* FROM Customers

WHERE CustomerName NOT LIKE 'a%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_not)

# WildCards

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [WildCards](https://dashboard.stige.in/index.php/topic/wildcards/)

In Progress

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the [LIKE](https://www.w3schools.com/sql/sql_like.asp) operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

### Wildcard Characters in MS Access

| **Symbol** | **Description** | **Example** |
| --- | --- | --- |
| \* | Represents zero or more characters | bl\* finds bl, black, blue, and blob |
| ? | Represents a single character | h?t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ! | Represents any character not in the brackets | h[!oa]t finds hit, but not hot and hat |
| – | Represents a range of characters | c[a-b]t finds cat and cbt |
| # | Represents any single numeric character | 2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295 |

### Wildcard Characters in SQL Server

| **Symbol** | **Description** | **Example** |
| --- | --- | --- |
| % | Represents zero or more characters | bl% finds bl, black, blue, and blob |
| \_ | Represents a single character | h\_t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ^ | Represents any character not in the brackets | h[^oa]t finds hit, but not hot and hat |
| – | Represents a range of characters | c[a-b]t finds cat and cbt |

All the wildcards can also be used in combinations!

Here are some examples showing different LIKE operators with ‘%’ and ‘\_’ wildcards:

All the wildcards can also be used in combinations!

Here are some examples showing different LIKE operators with ‘%’ and ‘\_’ wildcards:

| **LIKE Operator** | **Description** |
| --- | --- |
| WHERE CustomerName LIKE ‘a%’ | Finds any values that starts with “a” |
| WHERE CustomerName LIKE ‘%a’ | Finds any values that ends with “a” |
| WHERE CustomerName LIKE ‘%or%’ | Finds any values that have “or” in any position |
| WHERE CustomerName LIKE ‘\_r%’ | Finds any values that have “r” in the second position |
| WHERE CustomerName LIKE ‘a\_\_%’ | Finds any values that starts with “a” and are at least 3 characters in length |
| WHERE ContactName LIKE ‘a%o’ | Finds any values that starts with “a” and ends with “o” |

## Demo Database

The table below shows the complete “Customers” table from the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app’ | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |
| 11 | B’s Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 13 | Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. | 05022 | Mexico |
| 14 | Chop-suey Chinese | Yang Wang | Hauptstr. 29 | Bern | 3012 | Switzerland |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |
| 18 | Du monde entier | Janine Labrune | 67, rue des Cinquante Otages | Nantes | 44000 | France |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |
| 22 | FISSA Fabrica Inter. Salchichas S.A. | Diego Roel | C/ Moralzarzal, 86 | Madrid | 28034 | Spain |
| 23 | Folies gourmandes | Martine Rancé | 184, chaussée de Tournai | Lille | 59000 | France |
| 24 | Folk och fä HB | Maria Larsson | Åkergatan 24 | Bräcke | S-844 67 | Sweden |
| 25 | Frankenversand | Peter Franken | Berliner Platz 43 | München | 80805 | Germany |
| 26 | France restauration | Carine Schmitt | 54, rue Royale | Nantes | 44000 | France |
| 27 | Franchi S.p.A. | Paolo Accorti | Via Monte Bianco 34 | Torino | 10100 | Italy |
| 28 | Furia Bacalhau e Frutos do Mar | Lino Rodriguez | Jardim das rosas n. 32 | Lisboa | 1675 | Portugal |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 31 | Gourmet Lanchonetes | André Fonseca | Av. Brasil, 442 | Campinas | 04876-786 | Brazil |
| 32 | Great Lakes Food Market | Howard Snyder | 2732 Baker Blvd. | Eugene | 97403 | USA |
| 33 | GROSELLA-Restaurante | Manuel Pereira | 5ª Ave. Los Palos Grandes | Caracas | 1081 | Venezuela |
| 34 | Hanari Carnes | Mario Pontes | Rua do Paço, 67 | Rio de Janeiro | 05454-876 | Brazil |
| 35 | HILARIÓN-Abastos | Carlos Hernández | Carrera 22 con Ave. Carlos Soublette #8-35 | San Cristóbal | 5022 | Venezuela |
| 36 | Hungry Coyote Import Store | Yoshi Latimer | City Center Plaza 516 Main St. | Elgin | 97827 | USA |
| 37 | Hungry Owl All-Night Grocers | Patricia McKenna | 8 Johnstown Road | Cork |  | Ireland |
| 38 | Island Trading | Helen Bennett | Garden House Crowther Way | Cowes | PO31 7PJ | UK |
| 39 | Königlich Essen | Philip Cramer | Maubelstr. 90 | Brandenburg | 14776 | Germany |
| 40 | La corne d’abondance | Daniel Tonini | 67, avenue de l’Europe | Versailles | 78000 | France |
| 41 | La maison d’Asie | Annette Roulet | 1 rue Alsace-Lorraine | Toulouse | 31000 | France |
| 42 | Laughing Bacchus Wine Cellars | Yoshi Tannamuri | 1900 Oak St. | Vancouver | V3F 2K1 | Canada |
| 43 | Lazy K Kountry Store | John Steel | 12 Orchestra Terrace | Walla Walla | 99362 | USA |
| 44 | Lehmanns Marktstand | Renate Messner | Magazinweg 7 | Frankfurt a.M. | 60528 | Germany |
| 45 | Let’s Stop N Shop | Jaime Yorres | 87 Polk St. Suite 5 | San Francisco | 94117 | USA |
| 46 | LILA-Supermercado | Carlos González | Carrera 52 con Ave. Bolívar #65-98 Llano Largo | Barquisimeto | 3508 | Venezuela |
| 47 | LINO-Delicateses | Felipe Izquierdo | Ave. 5 de Mayo Porlamar | I. de Margarita | 4980 | Venezuela |
| 48 | Lonesome Pine Restaurant | Fran Wilson | 89 Chiaroscuro Rd. | Portland | 97219 | USA |
| 49 | Magazzini Alimentari Riuniti | Giovanni Rovelli | Via Ludovico il Moro 22 | Bergamo | 24100 | Italy |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium |
| 51 | Mère Paillarde | Jean Fresnière | 43 rue St. Laurent | Montréal | H1J 1C3 | Canada |
| 52 | Morgenstern Gesundkost | Alexander Feuer | Heerstr. 22 | Leipzig | 04179 | Germany |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina |
| 55 | Old World Delicatessen | Rene Phillips | 2743 Bering St. | Anchorage | 99508 | USA |
| 56 | Ottilies Käseladen | Henriette Pfalzheim | Mehrheimerstr. 369 | Köln | 50739 | Germany |
| 57 | Paris spécialités | Marie Bertrand | 265, boulevard Charonne | Paris | 75012 | France |
| 58 | Pericles Comidas clásicas | Guillermo Fernández | Calle Dr. Jorge Cash 321 | México D.F. | 05033 | Mexico |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 60 | Princesa Isabel Vinhoss | Isabel de Castro | Estrada da saúde n. 58 | Lisboa | 1756 | Portugal |
| 61 | Que Delícia | Bernardo Batista | Rua da Panificadora, 12 | Rio de Janeiro | 02389-673 | Brazil |
| 62 | Queen Cozinha | Lúcia Carvalho | Alameda dos Canàrios, 891 | São Paulo | 05487-020 | Brazil |
| 63 | QUICK-Stop | Horst Kloss | Taucherstraße 10 | Cunewalde | 01307 | Germany |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 65 | Rattlesnake Canyon Grocery | Paula Wilson | 2817 Milton Dr. | Albuquerque | 87110 | USA |
| 66 | Reggiani Caseifici | Maurizio Moroni | Strada Provinciale 124 | Reggio Emilia | 42100 | Italy |
| 67 | Ricardo Adocicados | Janete Limeira | Av. Copacabana, 267 | Rio de Janeiro | 02389-890 | Brazil |
| 68 | Richter Supermarkt | Michael Holz | Grenzacherweg 237 | Genève | 1203 | Switzerland |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |
| 70 | Santé Gourmet | Jonas Bergulfsen | Erling Skakkes gate 78 | Stavern | 4110 | Norway |
| 71 | Save-a-lot Markets | Jose Pavarotti | 187 Suffolk Ln. | Boise | 83720 | USA |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |
| 73 | Simons bistro | Jytte Petersen | Vinbæltet 34 | København | 1734 | Denmark |
| 74 | Spécialités du monde | Dominique Perrier | 25, rue Lauriston | Paris | 75016 | France |
| 75 | Split Rail Beer & Ale | Art Braunschweiger | P.O. Box 555 | Lander | 82520 | USA |
| 76 | Suprêmes délices | Pascale Cartrain | Boulevard Tirou, 255 | Charleroi | B-6000 | Belgium |
| 77 | The Big Cheese | Liz Nixon | 89 Jefferson Way Suite 2 | Portland | 97201 | USA |
| 78 | The Cracker Box | Liu Wong | 55 Grizzly Peak Rd. | Butte | 59801 | USA |
| 79 | Toms Spezialitäten | Karin Josephs | Luisenstr. 48 | Münster | 44087 | Germany |
| 80 | Tortuga Restaurante | Miguel Angel Paolino | Avda. Azteca 123 | México D.F. | 05033 | Mexico |
| 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil |
| 82 | Trail’s Head Gourmet Provisioners | Helvetius Nagy | 722 DaVinci Blvd. | Kirkland | 98034 | USA |
| 83 | Vaffeljernet | Palle Ibsen | Smagsløget 45 | Århus | 8200 | Denmark |
| 84 | Victuailles en stock | Mary Saveley | 2, rue du Commerce | Lyon | 69004 | France |
| 85 | Vins et alcools Chevalier | Paul Henriot | 59 rue de l’Abbaye | Reims | 51100 | France |
| 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |
| 89 | White Clover Markets | Karl Jablonski | 305 – 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

## Using the % Wildcard

The following SQL statement selects all customers with a City starting with “ber”:

### Example

SELECT \* FROM Customers

WHERE City LIKE 'ber%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_percent)

The following SQL statement selects all customers with a City containing the pattern “es”:

### Example

SELECT \* FROM Customers

WHERE City LIKE '%es%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_percent_pattern)

## Using the \_ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by “ondon”:

### Example

SELECT \* FROM Customers

WHERE City LIKE '\_ondon';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_underscore)

The following SQL statement selects all customers with a City starting with “L”, followed by any character, followed by “n”, followed by any character, followed by “on”:

### Example

SELECT \* FROM Customers

WHERE City LIKE 'L\_n\_on';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_underscore2)

## Using the [charlist] Wildcard

The following SQL statement selects all customers with a City starting with “b”, “s”, or “p”:

### Example

SELECT \* FROM Customers

WHERE City LIKE '[bsp]%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_charlist&ss=-1)

The following SQL statement selects all customers with a City starting with “a”, “b”, or “c”:

### Example

SELECT \* FROM Customers

WHERE City LIKE '[a-c]%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_charlist2&ss=-1)

## Using the [!charlist] Wildcard

The two following SQL statements select all customers with a City NOT starting with “b”, “s”, or “p”:

### Example

SELECT \* FROM Customers

WHERE City LIKE '[!bsp]%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_not_charlist&ss=-1)

Or:

### Example

SELECT \* FROM Customers

WHERE City NOT LIKE '[bsp]%';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_not_charlist2&ss=-1)

# In

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [In](https://dashboard.stige.in/index.php/topic/in/)

In Progress

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

### IN Syntax

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (value1, value2, ...);

or:

SELECT column\_name(s)

FROM table\_name

WHERE column\_name IN (SELECT STATEMENT);

## Demo Database

The table below shows the complete “Customers” table from the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app’ | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |
| 11 | B’s Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 13 | Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. | 05022 | Mexico |
| 14 | Chop-suey Chinese | Yang Wang | Hauptstr. 29 | Bern | 3012 | Switzerland |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |
| 18 | Du monde entier | Janine Labrune | 67, rue des Cinquante Otages | Nantes | 44000 | France |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |
| 22 | FISSA Fabrica Inter. Salchichas S.A. | Diego Roel | C/ Moralzarzal, 86 | Madrid | 28034 | Spain |
| 23 | Folies gourmandes | Martine Rancé | 184, chaussée de Tournai | Lille | 59000 | France |
| 24 | Folk och fä HB | Maria Larsson | Åkergatan 24 | Bräcke | S-844 67 | Sweden |
| 25 | Frankenversand | Peter Franken | Berliner Platz 43 | München | 80805 | Germany |
| 26 | France restauration | Carine Schmitt | 54, rue Royale | Nantes | 44000 | France |
| 27 | Franchi S.p.A. | Paolo Accorti | Via Monte Bianco 34 | Torino | 10100 | Italy |
| 28 | Furia Bacalhau e Frutos do Mar | Lino Rodriguez | Jardim das rosas n. 32 | Lisboa | 1675 | Portugal |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 31 | Gourmet Lanchonetes | André Fonseca | Av. Brasil, 442 | Campinas | 04876-786 | Brazil |
| 32 | Great Lakes Food Market | Howard Snyder | 2732 Baker Blvd. | Eugene | 97403 | USA |
| 33 | GROSELLA-Restaurante | Manuel Pereira | 5ª Ave. Los Palos Grandes | Caracas | 1081 | Venezuela |
| 34 | Hanari Carnes | Mario Pontes | Rua do Paço, 67 | Rio de Janeiro | 05454-876 | Brazil |
| 35 | HILARIÓN-Abastos | Carlos Hernández | Carrera 22 con Ave. Carlos Soublette #8-35 | San Cristóbal | 5022 | Venezuela |
| 36 | Hungry Coyote Import Store | Yoshi Latimer | City Center Plaza 516 Main St. | Elgin | 97827 | USA |
| 37 | Hungry Owl All-Night Grocers | Patricia McKenna | 8 Johnstown Road | Cork |  | Ireland |
| 38 | Island Trading | Helen Bennett | Garden House Crowther Way | Cowes | PO31 7PJ | UK |
| 39 | Königlich Essen | Philip Cramer | Maubelstr. 90 | Brandenburg | 14776 | Germany |
| 40 | La corne d’abondance | Daniel Tonini | 67, avenue de l’Europe | Versailles | 78000 | France |
| 41 | La maison d’Asie | Annette Roulet | 1 rue Alsace-Lorraine | Toulouse | 31000 | France |
| 42 | Laughing Bacchus Wine Cellars | Yoshi Tannamuri | 1900 Oak St. | Vancouver | V3F 2K1 | Canada |
| 43 | Lazy K Kountry Store | John Steel | 12 Orchestra Terrace | Walla Walla | 99362 | USA |
| 44 | Lehmanns Marktstand | Renate Messner | Magazinweg 7 | Frankfurt a.M. | 60528 | Germany |
| 45 | Let’s Stop N Shop | Jaime Yorres | 87 Polk St. Suite 5 | San Francisco | 94117 | USA |
| 46 | LILA-Supermercado | Carlos González | Carrera 52 con Ave. Bolívar #65-98 Llano Largo | Barquisimeto | 3508 | Venezuela |
| 47 | LINO-Delicateses | Felipe Izquierdo | Ave. 5 de Mayo Porlamar | I. de Margarita | 4980 | Venezuela |
| 48 | Lonesome Pine Restaurant | Fran Wilson | 89 Chiaroscuro Rd. | Portland | 97219 | USA |
| 49 | Magazzini Alimentari Riuniti | Giovanni Rovelli | Via Ludovico il Moro 22 | Bergamo | 24100 | Italy |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium |
| 51 | Mère Paillarde | Jean Fresnière | 43 rue St. Laurent | Montréal | H1J 1C3 | Canada |
| 52 | Morgenstern Gesundkost | Alexander Feuer | Heerstr. 22 | Leipzig | 04179 | Germany |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina |
| 55 | Old World Delicatessen | Rene Phillips | 2743 Bering St. | Anchorage | 99508 | USA |
| 56 | Ottilies Käseladen | Henriette Pfalzheim | Mehrheimerstr. 369 | Köln | 50739 | Germany |
| 57 | Paris spécialités | Marie Bertrand | 265, boulevard Charonne | Paris | 75012 | France |
| 58 | Pericles Comidas clásicas | Guillermo Fernández | Calle Dr. Jorge Cash 321 | México D.F. | 05033 | Mexico |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 60 | Princesa Isabel Vinhoss | Isabel de Castro | Estrada da saúde n. 58 | Lisboa | 1756 | Portugal |
| 61 | Que Delícia | Bernardo Batista | Rua da Panificadora, 12 | Rio de Janeiro | 02389-673 | Brazil |
| 62 | Queen Cozinha | Lúcia Carvalho | Alameda dos Canàrios, 891 | São Paulo | 05487-020 | Brazil |
| 63 | QUICK-Stop | Horst Kloss | Taucherstraße 10 | Cunewalde | 01307 | Germany |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 65 | Rattlesnake Canyon Grocery | Paula Wilson | 2817 Milton Dr. | Albuquerque | 87110 | USA |
| 66 | Reggiani Caseifici | Maurizio Moroni | Strada Provinciale 124 | Reggio Emilia | 42100 | Italy |
| 67 | Ricardo Adocicados | Janete Limeira | Av. Copacabana, 267 | Rio de Janeiro | 02389-890 | Brazil |
| 68 | Richter Supermarkt | Michael Holz | Grenzacherweg 237 | Genève | 1203 | Switzerland |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |
| 70 | Santé Gourmet | Jonas Bergulfsen | Erling Skakkes gate 78 | Stavern | 4110 | Norway |
| 71 | Save-a-lot Markets | Jose Pavarotti | 187 Suffolk Ln. | Boise | 83720 | USA |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |
| 73 | Simons bistro | Jytte Petersen | Vinbæltet 34 | København | 1734 | Denmark |
| 74 | Spécialités du monde | Dominique Perrier | 25, rue Lauriston | Paris | 75016 | France |
| 75 | Split Rail Beer & Ale | Art Braunschweiger | P.O. Box 555 | Lander | 82520 | USA |
| 76 | Suprêmes délices | Pascale Cartrain | Boulevard Tirou, 255 | Charleroi | B-6000 | Belgium |
| 77 | The Big Cheese | Liz Nixon | 89 Jefferson Way Suite 2 | Portland | 97201 | USA |
| 78 | The Cracker Box | Liu Wong | 55 Grizzly Peak Rd. | Butte | 59801 | USA |
| 79 | Toms Spezialitäten | Karin Josephs | Luisenstr. 48 | Münster | 44087 | Germany |
| 80 | Tortuga Restaurante | Miguel Angel Paolino | Avda. Azteca 123 | México D.F. | 05033 | Mexico |
| 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil |
| 82 | Trail’s Head Gourmet Provisioners | Helvetius Nagy | 722 DaVinci Blvd. | Kirkland | 98034 | USA |
| 83 | Vaffeljernet | Palle Ibsen | Smagsløget 45 | Århus | 8200 | Denmark |
| 84 | Victuailles en stock | Mary Saveley | 2, rue du Commerce | Lyon | 69004 | France |
| 85 | Vins et alcools Chevalier | Paul Henriot | 59 rue de l’Abbaye | Reims | 51100 | France |
| 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |
| 89 | White Clover Markets | Karl Jablonski | 305 – 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

## IN Operator Examples

The following SQL statement selects all customers that are located in “Germany”, “France” or “UK”:

### Example

SELECT \* FROM Customers

WHERE Country IN ('Germany', 'France', 'UK');

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_in)

The following SQL statement selects all customers that are NOT located in “Germany”, “France” or “UK”:

### Example

SELECT \* FROM Customers

WHERE Country NOT IN ('Germany', 'France', 'UK');

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_in_not)

The following SQL statement selects all customers that are from the same countries as the suppliers:

### Example

SELECT \* FROM Customers

WHERE Country IN (SELECT Country FROM Suppliers);

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_in2)

# Between

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Between](https://dashboard.stige.in/index.php/topic/between/)

In Progress

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

### BETWEEN Syntax

SELECT column\_name(s)

FROM table\_name

WHERE column\_name BETWEEN value1 AND value2;

## Demo Database

Below is a selection from the “Products” table in the Northwind sample database:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 – 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 – 550 ml bottles | 10 |
| 4 | Chef Anton’s Cajun Seasoning | 1 | 2 | 48 – 6 oz jars | 22 |
| 5 | Chef Anton’s Gumbo Mix | 1 | 2 | 36 boxes | 21.35 |

## BETWEEN Example

The following SQL statement selects all products with a price between 10 and 20:

#### Example

SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between)

## NOT BETWEEN Example

To display the products outside the range of the previous example, use NOT BETWEEN:

#### Example

SELECT \* FROM Products

WHERE Price NOT BETWEEN 10 AND 20;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_not_between)

## BETWEEN with IN Example

The following SQL statement selects all products with a price between 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3:

#### Example

SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20

AND CategoryID NOT IN (1,2,3);

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between_in)

## BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Mozzarella di Giovanni:

#### Example

SELECT \* FROM Products

WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'

ORDER BY ProductName;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between_text)

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Chef Anton’s Cajun Seasoning:

#### Example

SELECT \* FROM Products

WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning"

ORDER BY ProductName;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between_text2)

## NOT BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName not between Carnarvon Tigers and Mozzarella di Giovanni:

#### Example

SELECT \* FROM Products

WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'

ORDER BY ProductName;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_not_between_text)

## Sample Table

Below is a selection from the “Orders” table in the Northwind sample database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 7/4/1996 | 3 |
| 10249 | 81 | 6 | 7/5/1996 | 1 |
| 10250 | 34 | 4 | 7/8/1996 | 2 |
| 10251 | 84 | 3 | 7/9/1996 | 1 |
| 10252 | 76 | 4 | 7/10/1996 | 2 |

## BETWEEN Dates Example

The following SQL statement selects all orders with an OrderDate between ’01-July-1996′ and ’31-July-1996′:

#### Example

SELECT \* FROM Orders

WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between_date&ss=-1)

OR:

#### Example

SELECT \* FROM Orders

WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between_date2)

# Aliases

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Aliases](https://dashboard.stige.in/index.php/topic/aliases/)

In Progress

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

### Alias Column Syntax

SELECT column\_name AS alias\_name

FROM table\_name;

### Alias Table Syntax

SELECT column\_name(s)

FROM table\_name AS alias\_name;

## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the “Customers” table:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |

And a selection from the “Orders” table:

| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| --- | --- | --- | --- | --- |
| 10354 | 58 | 8 | 1996-11-14 | 3 |
| 10355 | 4 | 6 | 1996-11-15 | 1 |
| 10356 | 86 | 6 | 1996-11-18 | 2 |

## Alias for Columns Examples

The following SQL statement creates two aliases, one for the CustomerID column and one for the CustomerName column:

### Example

SELECT CustomerID AS ID, CustomerName AS Customer

FROM Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_column0)

The following SQL statement creates two aliases, one for the CustomerName column and one for the ContactName column. **Note:** It requires double quotation marks or square brackets if the alias name contains spaces:

### Example

SELECT CustomerName AS Customer, ContactName AS [Contact Person]

FROM Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_column)

The following SQL statement creates an alias named “Address” that combine four columns (Address, PostalCode, City and Country):

### Example

SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS Address

FROM Customers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_column2&ss=-1)

**Note:** To get the SQL statement above to work in MySQL use the following:

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address

FROM Customers;

## Alias for Tables Example

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the “Customers” and “Orders” tables, and give them the table aliases of “c” and “o” respectively (Here we use aliases to make the SQL shorter):

### Example

SELECT o.OrderID, o.OrderDate, c.CustomerName

FROM Customers AS c, Orders AS o

WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_table)

The following SQL statement is the same as above, but without aliases:

### Example

SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName

FROM Customers, Orders

WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_no)

Aliases can be useful when:

* There are more than one table involved in a query
* Functions are used in the query
* Column names are big or not very readable
* Two or more columns are combined together

# Union

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Union](https://dashboard.stige.in/index.php/topic/union/)

In Progress

The UNION operator is used to combine the result-set of two or more SELECT statements.

* Every SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in every SELECT statement must also be in the same order

### UNION Syntax

SELECT column\_name(s) FROM table1

UNION

SELECT column\_name(s) FROM table2;

### UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

SELECT column\_name(s) FROM table1

UNION ALL

SELECT column\_name(s) FROM table2;

**Note:** The column names in the result-set are usually equal to the column names in the first SELECT statement.

## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the “Customers” table:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexic |

And a selection from the “Suppliers” table:

| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly’s Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |

## SQL UNION Example

The following SQL statement returns the cities (only distinct values) from both the “Customers” and the “Suppliers” table:

### Example

SELECT City FROM Customers

UNION

SELECT City FROM Suppliers

ORDER BY City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union)

**Note:** If some customers or suppliers have the same city, each city will only be listed once, because UNION selects only distinct values. Use UNION ALL to also select duplicate values!

## SQL UNION ALL Example

The following SQL statement returns the cities (duplicate values also) from both the “Customers” and the “Suppliers” table:

### Example

SELECT City FROM Customers

UNION ALL

SELECT City FROM Suppliers

ORDER BY City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union_all)

## SQL UNION With WHERE

The following SQL statement returns the German cities (only distinct values) from both the “Customers” and the “Suppliers” table:

### Example

SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union2)

## SQL UNION ALL With WHERE

The following SQL statement returns the German cities (duplicate values also) from both the “Customers” and the “Suppliers” table:

### Example

SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION ALL

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union_all2)

## Another UNION Example

The following SQL statement lists all customers and suppliers:

### Example

SELECT 'Customer' AS Type, ContactName, City, Country

FROM Customers

UNION

SELECT 'Supplier', ContactName, City, Country

FROM Suppliers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union3)

# Exists

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Exists](https://dashboard.stige.in/index.php/topic/exists/)

In Progress

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

### EXISTS Syntax

SELECT column\_name(s)

FROM table\_name

WHERE EXISTS

(SELECT column\_name FROM table\_name WHERE condition);

## Demo Database

Below is a selection from the “Products” table in the Northwind sample database:

| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| --- | --- | --- | --- | --- | --- |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 – 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 – 550 ml bottles | 10 |
| 4 | Chef Anton’s Cajun Seasoning | 2 | 2 | 48 – 6 oz jars | 22 |
| 5 | Chef Anton’s Gumbo Mix | 2 | 2 | 36 boxes | 21.35 |

And a selection from the “Suppliers” table:

| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly’s Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |
| 4 | Tokyo Traders | Yoshi Nagase | 9-8 Sekimai Musashino-shi | Tokyo | 100 | Japan |

## SQL EXISTS Examples

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

### Example

SELECT SupplierName

FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_exists)

The following SQL statement returns TRUE and lists the suppliers with a product price equal to 22:

### Example

SELECT SupplierName

FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_exists2)

# Case Statement

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Case Statement](https://dashboard.stige.in/index.php/topic/case-statement/)

In Progress

## The SQL CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

## CASE Syntax

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

WHEN conditionN THEN resultN

ELSE result

END;

## Demo Database

Below is a selection from the “OrderDetails” table in the Northwind sample database:

| **OrderDetailID** | **OrderID** | **ProductID** | **Quantity** |
| --- | --- | --- | --- |
| 1 | 10248 | 11 | 12 |
| 2 | 10248 | 42 | 10 |
| 3 | 10248 | 72 | 5 |
| 4 | 10249 | 14 | 9 |
| 5 | 10249 | 51 | 40 |

## SQL CASE Examples

The following SQL goes through conditions and returns a value when the first condition is met:

### Example

SELECT OrderID, Quantity,

CASE

WHEN Quantity > 30 THEN 'The quantity is greater than 30'

WHEN Quantity = 30 THEN 'The quantity is 30'

ELSE 'The quantity is under 30'

END AS QuantityText

FROM OrderDetails;

[Try it yourself](https://www.w3schools.com/sql/trymysql.asp?filename=trysql_case)

The following SQL will order the customers by City. However, if City is NULL, then order by Country:

### Example

SELECT CustomerName, City, Country

FROM Customers

ORDER BY

(CASE

WHEN City IS NULL THEN Country

ELSE City

END);

[Try it yourself](https://www.w3schools.com/sql/trymysql.asp?filename=trysql_case2)

# Null Functions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 2: Basic Querying in SQL](https://dashboard.stige.in/index.php/lessons/task-2-basic-querying-in-sql/) [Null Functions](https://dashboard.stige.in/index.php/topic/null-functions/)

In Progress

## SQL IFNULL(), ISNULL(), COALESCE(), and NVL() Functions

Look at the following “Products” table:

| **P\_Id** | **ProductName** | **UnitPrice** | **UnitsInStock** | **UnitsOnOrder** |
| --- | --- | --- | --- | --- |
| 1 | Jarlsberg | 10.45 | 16 | 15 |
| 2 | Mascarpone | 32.56 | 23 |  |
| 3 | Gorgonzola | 15.67 | 9 | 20 |

Suppose that the “UnitsOnOrder” column is optional, and may contain NULL values.

Look at the following SELECT statement:

SELECT ProductName, UnitPrice \* (UnitsInStock + UnitsOnOrder)

FROM Products;

In the example above, if any of the “UnitsOnOrder” values are NULL, the result will be NULL.

## Solutions

**MySQL**

The MySQL [IFNULL()](https://www.w3schools.com/sql/func_mysql_ifnull.asp) function lets you return an alternative value if an expression is NULL:

SELECT ProductName, UnitPrice \* (UnitsInStock + IFNULL(UnitsOnOrder, 0))

FROM Products;

or we can use the [COALESCE()](https://www.w3schools.com/sql/func_mysql_coalesce.asp) function, like this:

SELECT ProductName, UnitPrice \* (UnitsInStock + COALESCE(UnitsOnOrder, 0))

FROM Products;

**SQL Server**

The SQL Server [ISNULL()](https://www.w3schools.com/sql/func_sqlserver_isnull.asp) function lets you return an alternative value when an expression is NULL:

SELECT ProductName, UnitPrice \* (UnitsInStock + ISNULL(UnitsOnOrder, 0))

FROM Products;

**MS Access**

The MS Access [IsNull()](https://www.w3schools.com/sql/func_msaccess_isnull.asp) function returns TRUE (-1) if the expression is a null value, otherwise FALSE (0):

SELECT ProductName, UnitPrice \* (UnitsInStock + IIF(IsNull(UnitsOnOrder), 0, UnitsOnOrder))

FROM Products;

**Oracle**

The Oracle NVL() function achieves the same result:

SELECT ProductName, UnitPrice \* (UnitsInStock + NVL(UnitsOnOrder, 0))

FROM Products;

# Task 3: Aggregate, Date and Sorting functions

# MIN() and MAX()

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Aggregate, Date and Sorting functions](https://dashboard.stige.in/index.php/lessons/task-3-aggregate-date-and-sorting-functions/) [MIN() and MAX()](https://dashboard.stige.in/index.php/topic/min-and-max/)

In Progress

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

### MIN() Syntax

SELECT MIN(column\_name)

FROM table\_name

WHERE condition;

### MAX() Syntax

SELECT MAX(column\_name)

FROM table\_name

WHERE condition;

## Demo Database

Below is a selection from the “Products” table in the Northwind sample database:

| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| --- | --- | --- | --- | --- | --- |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 – 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 – 550 ml bottles | 10 |
| 4 | Chef Anton’s Cajun Seasoning | 2 | 2 | 48 – 6 oz jars | 22 |
| 5 | Chef Anton’s Gumbo Mix | 2 | 2 | 36 boxes | 21.35 |

## MIN() Example

The following SQL statement finds the price of the cheapest product:

### Example

SELECT MIN(Price) AS SmallestPrice

FROM Products;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_min)

## MAX() Example

The following SQL statement finds the price of the most expensive product:

### Example

SELECT MAX(Price) AS LargestPrice

FROM Products;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_max)

# COUNT() , AVG() andd SUM()

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Aggregate, Date and Sorting functions](https://dashboard.stige.in/index.php/lessons/task-3-aggregate-date-and-sorting-functions/) [COUNT() , AVG() andd SUM()](https://dashboard.stige.in/index.php/topic/count-avg-andd-sum/)

In Progress

The COUNT() function returns the number of rows that matches a specified criterion.

### COUNT() Syntax

SELECT COUNT(column\_name)

FROM table\_name

WHERE condition;

The AVG() function returns the average value of a numeric column.

### AVG() Syntax

SELECT AVG(column\_name)

FROM table\_name

WHERE condition;

The SUM() function returns the total sum of a numeric column.

### SUM() Syntax

SELECT SUM(column\_name)

FROM table\_name

WHERE condition;

## Demo Database

Below is a selection from the “Products” table in the Northwind sample database:

| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| --- | --- | --- | --- | --- | --- |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 – 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 – 550 ml bottles | 10 |
| 4 | Chef Anton’s Cajun Seasoning | 2 | 2 | 48 – 6 oz jars | 22 |
| 5 | Chef Anton’s Gumbo Mix | 2 | 2 | 36 boxes | 21.35 |

## COUNT() Example

The following SQL statement finds the number of products:

### Example

SELECT COUNT(ProductID)

FROM Products;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_count)

**Note:** NULL values are not counted.

## AVG() Example

The following SQL statement finds the average price of all products:

### Example

SELECT AVG(Price)

FROM Products;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_avg)

**Note:** NULL values are ignored.

## Demo Database

Below is a selection from the “OrderDetails” table in the Northwind sample database:

| **OrderDetailID** | **OrderID** | **ProductID** | **Quantity** |
| --- | --- | --- | --- |
| 1 | 10248 | 11 | 12 |
| 2 | 10248 | 42 | 10 |
| 3 | 10248 | 72 | 5 |
| 4 | 10249 | 14 | 9 |
| 5 | 10249 | 51 | 40 |

## SUM() Example

The following SQL statement finds the sum of the “Quantity” fields in the “OrderDetails” table:

### Example

SELECT SUM(Quantity)

FROM OrderDetails;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_sum)

**Note:** NULL values are ignored.

# More Aggregate Function

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Aggregate, Date and Sorting functions](https://dashboard.stige.in/index.php/lessons/task-3-aggregate-date-and-sorting-functions/) [More Aggregate Function](https://dashboard.stige.in/index.php/topic/more-aggregate-function/)

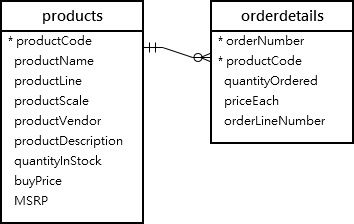
In Progress

MySQL supports the following aggregate functions:

| **Aggregate function** | **Description** |
| --- | --- |
| [AVG()](https://www.mysqltutorial.org/mysql-avg/) | Return the average of non-NULL values. |
| BIT\_AND() | Return bitwise AND. |
| BIT\_OR() | Return bitwise OR. |
| BIT\_XOR() | Return bitwise XOR. |
| [COUNT()](https://www.mysqltutorial.org/mysql-count/) | Return the number of rows in a group, including rows with NULL values. |
| [GROUP\_CONCAT()](https://www.mysqltutorial.org/mysql-group_concat/) | Return a concatenated string. |
| JSON\_ARRAYAGG() | Return result set as a single JSON array. |
| JSON\_OBJECTAGG() | Return result set as a single JSON object. |
| [MAX()](https://www.mysqltutorial.org/mysql-max-function/) | Return the highest value (maximum) in a set of non-NULL values. |
| [MIN()](https://www.mysqltutorial.org/mysql-min/) | Return the lowest value (minimum) in a set of non-NULL values. |
| [STDEV()](https://www.mysqltutorial.org/mysql-standard-deviation/) | Return the population standard deviation. |
| STDDEV\_POP() | Return the population standard deviation. |
| STDDEV\_SAMP() | Return the sample standard deviation. |
| [SUM()](https://www.mysqltutorial.org/mysql-sum/) | Return the summation of all non-NULL values a set. |
| VAR\_POP() | Return the population standard variance. |
| VARP\_SAM() | Return the sample variance. |
| VARIANCE() | Return the population standard variance. |

## MySQL aggregate function examples

We will use the products and orderdetails tables from the [sample database](https://www.mysqltutorial.org/mysql-sample-database.aspx) for demonstration:



### MySQL aggregate function – MAX() function examples

The [MAX()](https://www.mysqltutorial.org/mysql-max-function/) function returns the maximum value in a set.

The following statement uses the MAX() function with the GROUP BY clause to get the highest price per product line:

SELECT

productLine, MAX(buyPrice)

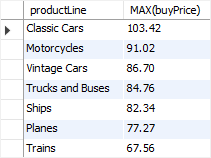
FROM

products

GROUP BY productLine

ORDER BY MAX(buyPrice) DESC;

[Try it yourself](https://www.mysqltutorial.org/tryit/query/mysql-aggregate-functions/#1)



### MySQL aggregate function – MIN() function examples

The [MIN()](https://www.mysqltutorial.org/mysql-min/) function returns the minimum value in a set of values.

The following example uses the MIN() function with the GROUP BY clause to get the lowest price per product line:

SELECT

productLine,

MIN(buyPrice)

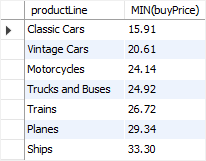
FROM

products

GROUP BY productLine

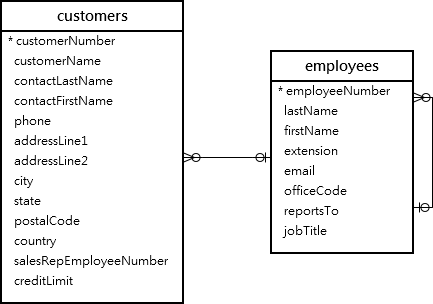
ORDER BY MIN(buy

[Try it yourself](https://www.mysqltutorial.org/tryit/query/mysql-aggregate-functions/#1)



### MySQL aggregate function – GROUP\_CONCAT() function example

The [GROUP\_CONCAT()](https://www.mysqltutorial.org/mysql-group_concat/) concatenates a set of strings and returns the concatenated string. See the following employees and customers tables:



The following statement uses the GROUP\_CONCAT() function to return the sales staffs and list of customers that each sales staff is in charge of:

SELECT

firstName,

lastName,

GROUP\_CONCAT(

DISTINCT customername

ORDER BY customerName) customers

FROM

employees

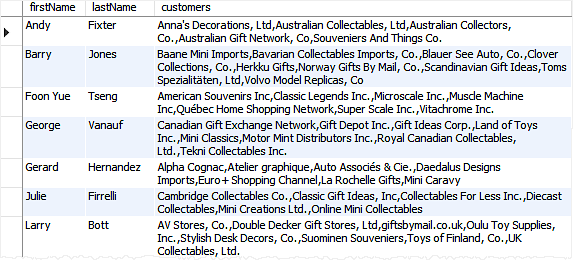
INNER JOIN customers

ON customers.salesRepEmployeeNumber = employeeNumber

GROUP BY employeeNumber

ORDER BY firstName , lastname;

[Try it yourself](https://www.mysqltutorial.org/tryit/query/mysql-aggregate-functions/#1)



# GROUP BY

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Aggregate, Date and Sorting functions](https://dashboard.stige.in/index.php/lessons/task-3-aggregate-date-and-sorting-functions/) [GROUP BY](https://dashboard.stige.in/index.php/topic/group-by/)

In Progress

The GROUP BY statement groups rows that have the same values into summary rows, like “find the number of customers in each country”.

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

### GROUP BY Syntax

SELECT column\_name(s)

FROM table\_name

WHERE condition

GROUP BY column\_name(s)

ORDER BY column\_name(s);

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

### Example

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_groupby)

The following SQL statement lists the number of customers in each country, sorted high to low:

### Example

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

ORDER BY COUNT(CustomerID) DESC;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_groupby_orderby)

## Demo Database

Below is a selection from the “Orders” table in the Northwind sample database:

| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| --- | --- | --- | --- | --- |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the “Shippers” table:

| **ShipperID** | **ShipperName** |
| --- | --- |
| 1 | Speedy Express |
| 2 | United Package |
| 3 | Federal Shipping |

## GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper:

### Example

SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders

LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID

GROUP BY ShipperName;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_groupby1)

# HAVING Clause

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Aggregate, Date and Sorting functions](https://dashboard.stige.in/index.php/lessons/task-3-aggregate-date-and-sorting-functions/) [HAVING Clause](https://dashboard.stige.in/index.php/topic/having-clause/)

In Progress

## The SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

### HAVING Syntax

SELECT column\_name(s)

FROM table\_name

WHERE condition

GROUP BY column\_name(s)

HAVING condition

ORDER BY column\_name(s);

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

### Example

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_having)

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

### Example

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5

ORDER BY COUNT(CustomerID) DESC;

[Try it yourself](https://dashboard.stige.in/index.php/topic/having-clause/)

## Demo Database

Below is a selection from the “Orders” table in the Northwind sample database:

| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| --- | --- | --- | --- | --- |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the “Employees” table:

| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| 1 | Davolio | Nancy | 1968-12-08 | EmpID1.pic | Education includes a BA…. |
| 2 | Fuller | Andrew | 1952-02-19 | EmpID2.pic | Andrew received his BTS…. |
| 3 | Leverling | Janet | 1963-08-30 | EmpID3.pic | Janet has a BS degree…. |

## More HAVING Examples

The following SQL statement lists the employees that have registered more than 10 orders:

### Example

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders

FROM (Orders

INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)

GROUP BY LastName

HAVING COUNT(Orders.OrderID) > 10;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_having2)

The following SQL statement lists if the employees “Davolio” or “Fuller” have registered more than 25 orders:

### Example

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders

FROM Orders

INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID

WHERE LastName = 'Davolio' OR LastName = 'Fuller'

GROUP BY LastName

HAVING COUNT(Orders.OrderID) > 25;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_having_where)

# ORDER BY

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Aggregate, Date and Sorting functions](https://dashboard.stige.in/index.php/lessons/task-3-aggregate-date-and-sorting-functions/) [ORDER BY](https://dashboard.stige.in/index.php/topic/order-by/)

In Progress

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

### ORDER BY Syntax

SELECT column1, column2, ...

FROM table\_name

ORDER BY column1, column2, ... ASC|DESC;

## Demo Database

Below is a selection from the “Customers” table in the Northwind sample database:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## ORDER BY Example

The following SQL statement selects all customers from the “Customers” table, sorted by the “Country” column:

### Example

SELECT \* FROM Customers

ORDER BY Country;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_orderby)

## ORDER BY DESC Example

The following SQL statement selects all customers from the “Customers” table, sorted DESCENDING by the “Country” column:

### Example

SELECT \* FROM Customers

ORDER BY Country DESC;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_orderby_desc)

## ORDER BY Several Columns Example

The following SQL statement selects all customers from the “Customers” table, sorted by the “Country” and the “CustomerName” column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

### Example

SELECT \* FROM Customers

ORDER BY Country, CustomerName;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_orderby2)

## ORDER BY Several Columns Example 2

The following SQL statement selects all customers from the “Customers” table, sorted ascending by the “Country” and descending by the “CustomerName” column:

### Example

SELECT \* FROM Customers

ORDER BY Country ASC, CustomerName DESC;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_orderby3)

# SQL SERVER FUNCTIONS

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Aggregate, Date and Sorting functions](https://dashboard.stige.in/index.php/lessons/task-3-aggregate-date-and-sorting-functions/) [SQL SERVER FUNCTIONS](https://dashboard.stige.in/index.php/topic/sql-server-functions/)

In Progress

SQL Server has many built-in functions.

This reference contains string, numeric, date, conversion, and some advanced functions in SQL Server.

## SQL Server String Functions

| **Function** | **Description** |
| --- | --- |
| [ASCII](https://www.w3schools.com/sql/func_sqlserver_ascii.asp) | Returns the ASCII value for the specific character |
| [CHAR](https://www.w3schools.com/sql/func_sqlserver_char.asp) | Returns the character based on the ASCII code |
| [CHARINDEX](https://www.w3schools.com/sql/func_sqlserver_charindex.asp) | Returns the position of a substring in a string |
| [CONCAT](https://www.w3schools.com/sql/func_sqlserver_concat.asp) | Adds two or more strings together |
| [Concat with +](https://www.w3schools.com/sql/func_sqlserver_concat_with_plus.asp) | Adds two or more strings together |
| [CONCAT\_WS](https://www.w3schools.com/sql/func_sqlserver_concat_ws.asp) | Adds two or more strings together with a separator |
| [DATALENGTH](https://www.w3schools.com/sql/func_sqlserver_datalength.asp) | Returns the number of bytes used to represent an expression |
| [DIFFERENCE](https://www.w3schools.com/sql/func_sqlserver_difference.asp) | Compares two SOUNDEX values, and returns an integer value |
| [FORMAT](https://www.w3schools.com/sql/func_sqlserver_format.asp) | Formats a value with the specified format |
| [LEFT](https://www.w3schools.com/sql/func_sqlserver_left.asp) | Extracts a number of characters from a string (starting from left) |
| [LEN](https://www.w3schools.com/sql/func_sqlserver_len.asp) | Returns the length of a string |
| [LOWER](https://www.w3schools.com/sql/func_sqlserver_lower.asp) | Converts a string to lower-case |
| [LTRIM](https://www.w3schools.com/sql/func_sqlserver_ltrim.asp) | Removes leading spaces from a string |
| [NCHAR](https://www.w3schools.com/sql/func_sqlserver_nchar.asp) | Returns the Unicode character based on the number code |
| [PATINDEX](https://www.w3schools.com/sql/func_sqlserver_patindex.asp) | Returns the position of a pattern in a string |
| [QUOTENAME](https://www.w3schools.com/sql/func_sqlserver_quotename.asp) | Returns a Unicode string with delimiters added to make the string a valid SQL Server delimited identifier |
| [REPLACE](https://www.w3schools.com/sql/func_sqlserver_replace.asp) | Replaces all occurrences of a substring within a string, with a new substring |
| [REPLICATE](https://www.w3schools.com/sql/func_sqlserver_replicate.asp) | Repeats a string a specified number of times |
| [REVERSE](https://www.w3schools.com/sql/func_sqlserver_reverse.asp) | Reverses a string and returns the result |
| [RIGHT](https://www.w3schools.com/sql/func_sqlserver_right.asp) | Extracts a number of characters from a string (starting from right) |
| [RTRIM](https://www.w3schools.com/sql/func_sqlserver_rtrim.asp) | Removes trailing spaces from a string |
| [SOUNDEX](https://www.w3schools.com/sql/func_sqlserver_soundex.asp) | Returns a four-character code to evaluate the similarity of two strings |
| [SPACE](https://www.w3schools.com/sql/func_sqlserver_space.asp) | Returns a string of the specified number of space characters |
| [STR](https://www.w3schools.com/sql/func_sqlserver_str.asp) | Returns a number as string |
| [STUFF](https://www.w3schools.com/sql/func_sqlserver_stuff.asp) | Deletes a part of a string and then inserts another part into the string, starting at a specified position |
| [SUBSTRING](https://www.w3schools.com/sql/func_sqlserver_substring.asp) | Extracts some characters from a string |
| [TRANSLATE](https://www.w3schools.com/sql/func_sqlserver_translate.asp) | Returns the string from the first argument after the characters specified in the second argument are translated into the characters specified in the third argument. |
| [TRIM](https://www.w3schools.com/sql/func_sqlserver_trim.asp) | Removes leading and trailing spaces (or other specified characters) from a string |
| [UNICODE](https://www.w3schools.com/sql/func_sqlserver_unicode.asp) | Returns the Unicode value for the first character of the input expression |
| [UPPER](https://www.w3schools.com/sql/func_sqlserver_upper.asp) | Converts a string to upper-case |

## SQL Server Math/Numeric Functions

| **Function** | **Description** |
| --- | --- |
| [ABS](https://www.w3schools.com/sql/func_sqlserver_abs.asp) | Returns the absolute value of a number |
| [ACOS](https://www.w3schools.com/sql/func_sqlserver_acos.asp) | Returns the arc cosine of a number |
| [ASIN](https://www.w3schools.com/sql/func_sqlserver_asin.asp) | Returns the arc sine of a number |
| [ATAN](https://www.w3schools.com/sql/func_sqlserver_atan.asp) | Returns the arc tangent of a number |
| [ATN2](https://www.w3schools.com/sql/func_sqlserver_atn2.asp) | Returns the arc tangent of two numbers |
| [AVG](https://www.w3schools.com/sql/func_sqlserver_avg.asp) | Returns the average value of an expression |
| [CEILING](https://www.w3schools.com/sql/func_sqlserver_ceiling.asp) | Returns the smallest integer value that is >= a number |
| [COUNT](https://www.w3schools.com/sql/func_sqlserver_count.asp) | Returns the number of records returned by a select query |
| [COS](https://www.w3schools.com/sql/func_sqlserver_cos.asp) | Returns the cosine of a number |
| [COT](https://www.w3schools.com/sql/func_sqlserver_cot.asp) | Returns the cotangent of a number |
| [DEGREES](https://www.w3schools.com/sql/func_sqlserver_degrees.asp) | Converts a value in radians to degrees |
| [EXP](https://www.w3schools.com/sql/func_sqlserver_exp.asp) | Returns e raised to the power of a specified number |
| [FLOOR](https://www.w3schools.com/sql/func_sqlserver_floor.asp) | Returns the largest integer value that is <= to a number |
| [LOG](https://www.w3schools.com/sql/func_sqlserver_log.asp) | Returns the natural logarithm of a number, or the logarithm of a number to a specified base |
| [LOG10](https://www.w3schools.com/sql/func_sqlserver_log10.asp) | Returns the natural logarithm of a number to base 10 |
| [MAX](https://www.w3schools.com/sql/func_sqlserver_max.asp) | Returns the maximum value in a set of values |
| [MIN](https://www.w3schools.com/sql/func_sqlserver_min.asp) | Returns the minimum value in a set of values |
| [PI](https://www.w3schools.com/sql/func_sqlserver_pi.asp) | Returns the value of PI |
| [POWER](https://www.w3schools.com/sql/func_sqlserver_power.asp) | Returns the value of a number raised to the power of another number |
| [RADIANS](https://www.w3schools.com/sql/func_sqlserver_radians.asp) | Converts a degree value into radians |
| [RAND](https://www.w3schools.com/sql/func_sqlserver_rand.asp) | Returns a random number |
| [ROUND](https://www.w3schools.com/sql/func_sqlserver_round.asp) | Rounds a number to a specified number of decimal places |
| [SIGN](https://www.w3schools.com/sql/func_sqlserver_sign.asp) | Returns the sign of a number |
| [SIN](https://www.w3schools.com/sql/func_sqlserver_sin.asp) | Returns the sine of a number |
| [SQRT](https://www.w3schools.com/sql/func_sqlserver_sqrt.asp) | Returns the square root of a number |
| [SQUARE](https://www.w3schools.com/sql/func_sqlserver_square.asp) | Returns the square of a number |
| [SUM](https://www.w3schools.com/sql/func_sqlserver_sum.asp) | Calculates the sum of a set of values |
| [TAN](https://www.w3schools.com/sql/func_sqlserver_tan.asp) | Returns the tangent of a number |

## SQL Server Date Functions

| **Function** | **Description** |
| --- | --- |
| [CURRENT\_TIMESTAMP](https://www.w3schools.com/sql/func_sqlserver_current_timestamp.asp) | Returns the current date and time |
| [DATEADD](https://www.w3schools.com/sql/func_sqlserver_dateadd.asp) | Adds a time/date interval to a date and then returns the date |
| [DATEDIFF](https://www.w3schools.com/sql/func_sqlserver_datediff.asp) | Returns the difference between two dates |
| [DATEFROMPARTS](https://www.w3schools.com/sql/func_sqlserver_datefromparts.asp) | Returns a date from the specified parts (year, month, and day values) |
| [DATENAME](https://www.w3schools.com/sql/func_sqlserver_datename.asp) | Returns a specified part of a date (as string) |
| [DATEPART](https://www.w3schools.com/sql/func_sqlserver_datepart.asp) | Returns a specified part of a date (as integer) |
| [DAY](https://www.w3schools.com/sql/func_sqlserver_day.asp) | Returns the day of the month for a specified date |
| [GETDATE](https://www.w3schools.com/sql/func_sqlserver_getdate.asp) | Returns the current database system date and time |
| [GETUTCDATE](https://www.w3schools.com/sql/func_sqlserver_getutcdate.asp) | Returns the current database system UTC date and time |
| [ISDATE](https://www.w3schools.com/sql/func_sqlserver_isdate.asp) | Checks an expression and returns 1 if it is a valid date, otherwise 0 |
| [MONTH](https://www.w3schools.com/sql/func_sqlserver_month.asp) | Returns the month part for a specified date (a number from 1 to 12) |
| [SYSDATETIME](https://www.w3schools.com/sql/func_sqlserver_sysdatetime.asp) | Returns the date and time of the SQL Server |
| [YEAR](https://www.w3schools.com/sql/func_sqlserver_year.asp) | Returns the year part for a specified date |

## SQL Server Advanced Functions

| **Function** | **Description** |
| --- | --- |
| [CAST](https://www.w3schools.com/sql/func_sqlserver_cast.asp) | Converts a value (of any type) into a specified datatype |
| [COALESCE](https://www.w3schools.com/sql/func_sqlserver_coalesce.asp) | Returns the first non-null value in a list |
| [CONVERT](https://www.w3schools.com/sql/func_sqlserver_convert.asp) | Converts a value (of any type) into a specified datatype |
| [CURRENT\_USER](https://www.w3schools.com/sql/func_sqlserver_current_user.asp) | Returns the name of the current user in the SQL Server database |
| [IIF](https://www.w3schools.com/sql/func_sqlserver_iif.asp) | Returns a value if a condition is TRUE, or another value if a condition is FALSE |
| [ISNULL](https://www.w3schools.com/sql/func_sqlserver_isnull.asp) | Return a specified value if the expression is NULL, otherwise return the expression |
| [ISNUMERIC](https://www.w3schools.com/sql/func_sqlserver_isnumeric.asp) | Tests whether an expression is numeric |
| [NULLIF](https://www.w3schools.com/sql/func_sqlserver_nullif.asp) | Returns NULL if two expressions are equal |
| [SESSION\_USER](https://www.w3schools.com/sql/func_sqlserver_session_user.asp) | Returns the name of the current user in the SQL Server database |
| [SESSIONPROPERTY](https://www.w3schools.com/sql/func_sqlserver_sessionproperty.asp) | Returns the session settings for a specified option |
| [SYSTEM\_USER](https://www.w3schools.com/sql/func_sqlserver_system_user.asp) | Returns the login name for the current user |
| [USER\_NAME](https://www.w3schools.com/sql/func_sqlserver_user_name.asp) | Returns the database user name based on the specified id |

# Working with Dates

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 3: Aggregate, Date and Sorting functions](https://dashboard.stige.in/index.php/lessons/task-3-aggregate-date-and-sorting-functions/) [Working with Dates](https://dashboard.stige.in/index.php/topic/working-with-dates/)

In Progress

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

## SQL Date Data Types

**MySQL** comes with the following data types for storing a date or a date/time value in the database:

* DATE – format YYYY-MM-DD
* DATETIME – format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP – format: YYYY-MM-DD HH:MI:SS
* YEAR – format YYYY or YY

**SQL Server** comes with the following data types for storing a date or a date/time value in the database:

* DATE – format YYYY-MM-DD
* DATETIME – format: YYYY-MM-DD HH:MI:SS
* SMALLDATETIME – format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP – format: a unique number

**Note:** The date types are chosen for a column when you create a new table in your database!

## SQL Working with Dates

Look at the following table:

### Orders Table

| **OrderId** | **ProductName** | **OrderDate** |
| --- | --- | --- |
| 1 | Geitost | 2008-11-11 |
| 2 | Camembert Pierrot | 2008-11-09 |
| 3 | Mozzarella di Giovanni | 2008-11-11 |
| 4 | Mascarpone Fabioli | 2008-10-29 |

Now we want to select the records with an OrderDate of “2008-11-11” from the table above.

We use the following SELECT statement:

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

he result-set will look like this:

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | Geitost | 2008-11-11 |
| 3 | Mozzarella di Giovanni | 2008-11-11 |

**Note:** Two dates can easily be compared if there is no time component involved!

Now, assume that the “Orders” table looks like this (notice the added time-component in the “OrderDate” column):

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | Geitost | 2008-11-11 13:23:44 |
| 2 | Camembert Pierrot | 2008-11-09 15:45:21 |
| 3 | Mozzarella di Giovanni | 2008-11-11 11:12:01 |
| 4 | Mascarpone Fabioli | 2008-10-29 14:56:59 |

If we use the same SELECT statement as above:

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

we will get no result! This is because the query is looking only for dates with no time portion.

**Tip:** To keep your queries simple and easy to maintain, do not use time-components in your dates, unless you have to!

# Task 4: Joins and Set Operations

# JOINS

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [JOINS](https://dashboard.stige.in/index.php/topic/joins/)

In Progress

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let’s look at a selection from the “Orders” table:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerID** | **OrderDate** |
| 10308 | 2 | 1996-09-18 |
| 10309 | 37 | 1996-09-19 |
| 10310 | 77 | 1996-09-20 |

Then, look at a selection from the “Customers” table:

| **CustomerID** | **CustomerName** | **ContactName** | **Country** |
| --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

Notice that the “CustomerID” column in the “Orders” table refers to the “CustomerID” in the “Customers” table. The relationship between the two tables above is the “CustomerID” column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

## Example

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join)

and it will produce something like this:

| **OrderID** | **CustomerName** | **OrderDate** |
| --- | --- | --- |
| 10308 | Ana Trujillo Emparedados y helados | 9/18/1996 |
| 10365 | Antonio Moreno Taquería | 11/27/1996 |
| 10383 | Around the Horn | 12/16/1996 |
| 10355 | Around the Horn | 11/15/1996 |
| 10278 | Berglunds snabbköp | 8/12/1996 |

# SQL INNER JOIN

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [SQL INNER JOIN](https://dashboard.stige.in/index.php/topic/sql-inner-join/)

In Progress

The INNER JOIN keyword selects records that have matching values in both tables.

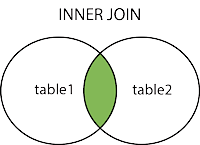
### INNER JOIN Syntax

SELECT column\_name(s)

FROM table1

INNER JOIN table2

ON table1.column\_name = table2.column\_name;



### Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the “Orders” table:

| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| --- | --- | --- | --- | --- |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

And a selection from the “Customers” table:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

## SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

### Example

SELECT Orders.OrderID, Customers.CustomerName

FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join_inner)

**Note:** The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the “Orders” table that do not have matches in “Customers”, these orders will not be shown!

## JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

### Example

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName

FROM ((Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)

INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join_inner2)

# SQL LEFT JOIN

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [SQL LEFT JOIN](https://dashboard.stige.in/index.php/topic/sql-left-join/)

In Progress

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

### LEFT JOIN Syntax

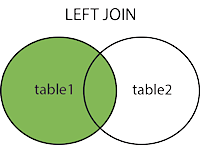
SELECT column\_name(s)

FROM table1

LEFT JOIN table2

ON table1.column\_name = table2.column\_name;

**Note:** In some databases LEFT JOIN is called LEFT OUTER JOIN.



## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the “Customers” table:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the “Orders” table:

| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| --- | --- | --- | --- | --- |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

## SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

## Example

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join_left)

**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

# SQL RIGHT JOIN

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [SQL RIGHT JOIN](https://dashboard.stige.in/index.php/topic/sql-right-join/)

In Progress

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

### RIGHT JOIN Syntax

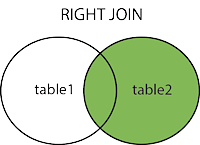
SELECT column\_name(s)

FROM table1

RIGHT JOIN table2

ON table1.column\_name = table2.column\_name;

**Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the “Orders” table:

| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| --- | --- | --- | --- | --- |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

And a selection from the “Employees” table:

| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** |
| --- | --- | --- | --- | --- |
| 1 | Davolio | Nancy | 12/8/1968 | EmpID1.pic |
| 2 | Fuller | Andrew | 2/19/1952 | EmpID2.pic |
| 3 | Leverling | Janet | 8/30/1963 | EmpID3.pic |

## SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have placed:

### Example

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

FROM Orders

RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID

ORDER BY Orders.OrderID;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join_right&ss=-1)

# SQL FULL OUTER JOIN

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [SQL FULL OUTER JOIN](https://dashboard.stige.in/index.php/topic/sql-full-outer-join/)

In Progress

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

**Tip:** FULL OUTER JOIN and FULL JOIN are the same.

### FULL OUTER JOIN Syntax

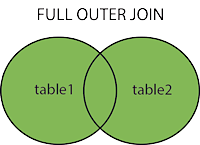
SELECT column\_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column\_name = table2.column\_name

WHERE condition;



**Note:** FULL OUTER JOIN can potentially return very large result-sets!

## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the “Customers” table:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the “Orders” table:

| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| --- | --- | --- | --- | --- |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

## SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID

ORDER BY Customers.CustomerName;

A selection from the result set may look like this:

| **CustomerName** | **OrderID** |
| --- | --- |
| Alfreds Futterkiste | Null |
| Ana Trujillo Emparedados y helados | 10308 |
| Antonio Moreno Taquería | Null |

**Note:** The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in “Customers” that do not have matches in “Orders”, or if there are rows in “Orders” that do not have matches in “Customers”, those rows will be listed as well.

# SQL Self JOIN

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [SQL Self JOIN](https://dashboard.stige.in/index.php/topic/sql-self-join/)

In Progress

A self join is a regular join, but the table is joined with itself.

### Self Join Syntax

SELECT column\_name(s)

FROM table1 T1, table1 T2

WHERE condition;

T1 and T2 are different table aliases for the same table.

## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the “Customers” table:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

## SQL Self Join Example

The following SQL statement matches customers that are from the same city:

SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City

FROM Customers A, Customers B

WHERE A.CustomerID <> B.CustomerID

AND A.City = B.City

ORDER BY A.City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join_self)

# Cross JOIN

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [Cross JOIN](https://dashboard.stige.in/index.php/topic/cross-join/)

In Progress

he SQL CROSS JOIN produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table if no WHERE clause is used along with CROSS JOIN.This kind of result is called as Cartesian Product.

If WHERE clause is used with CROSS JOIN, it functions like an INNER JOIN.

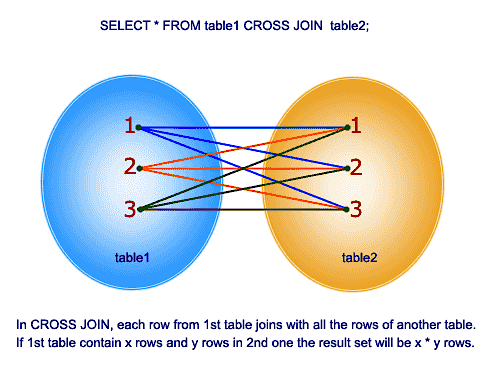
An alternative way of achieving the same result is to use column names separated by commas after SELECT and mentioning the table names involved, after a FROM clause.

**Syntax:**

SELECT \*

FROM table1

CROSS JOIN table2;



**Example:**

Here is an example of cross join in SQL between two tables.

**Sample table: foods**

+---------+--------------+-----------+------------+

| ITEM\_ID | ITEM\_NAME | ITEM\_UNIT | COMPANY\_ID |

+---------+--------------+-----------+------------+

| 1 | Chex Mix | Pcs | 16 |

| 6 | Cheez-It | Pcs | 15 |

| 2 | BN Biscuit | Pcs | 15 |

| 3 | Mighty Munch | Pcs | 17 |

| 4 | Pot Rice | Pcs | 15 |

| 5 | Jaffa Cakes | Pcs | 18 |

| 7 | Salt n Shake | Pcs | |

+---------+--------------+-----------+------------+

**Sample table: company**

+------------+---------------+--------------+

| COMPANY\_ID | COMPANY\_NAME | COMPANY\_CITY |

+------------+---------------+--------------+

| 18 | Order All | Boston |

| 15 | Jack Hill Ltd | London |

| 16 | Akas Foods | Delhi |

| 17 | Foodies. | London |

| 19 | sip-n-Bite. | New York |

+------------+---------------+--------------+

To get item name and item unit columns from foods table and company name, company city columns from company table, after a CROSS JOINING with these mentioned tables, the following SQL statement can be used:

**SQL Code:**

SELECT foods.item\_name,foods.item\_unit,

company.company\_name,company.company\_city

FROM foods

CROSS JOIN company;

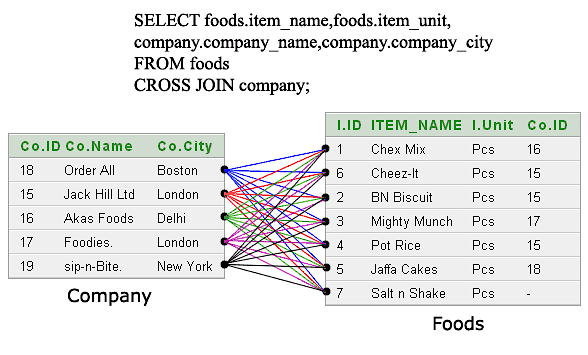
or

**SQL Code:**

SELECT foods.item\_name,foods.item\_unit,

company.company\_name,company.company\_city

FROM foods,company;



Output:

ITEM\_NAME ITEM\_UNIT COMPANY\_NAME COMPANY\_CITY

--------------- ---------- --------------- ---------------

Chex Mix Pcs Order All Boston

Cheez-It Pcs Order All Boston

BN Biscuit Pcs Order All Boston

Mighty Munch Pcs Order All Boston

Pot Rice Pcs Order All Boston

Jaffa Cakes Pcs Order All Boston

Salt n Shake Pcs Order All Boston

Chex Mix Pcs Jack Hill Ltd London

Cheez-It Pcs Jack Hill Ltd London

BN Biscuit Pcs Jack Hill Ltd London

Mighty Munch Pcs Jack Hill Ltd London

Pot Rice Pcs Jack Hill Ltd London

Jaffa Cakes Pcs Jack Hill Ltd London

Salt n Shake Pcs Jack Hill Ltd London

Chex Mix Pcs Akas Foods Delhi

Cheez-It Pcs Akas Foods Delhi

BN Biscuit Pcs Akas Foods Delhi

Mighty Munch Pcs Akas Foods Delhi

Pot Rice Pcs Akas Foods Delhi

Jaffa Cakes Pcs Akas Foods Delhi

Salt n Shake Pcs Akas Foods Delhi

Chex Mix Pcs Foodies. London

.........

.........

**More presentaion of the said output:**



# More on Cross JOIN

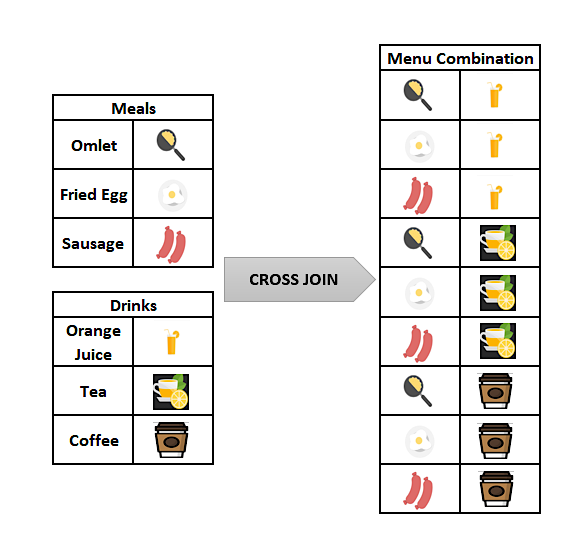
[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [More on Cross JOIN](https://dashboard.stige.in/index.php/topic/more-on-cross-join/)

In Progress

The CROSS JOIN is used to generate a paired combination of each row of the first table with each row of the second table. This join type is also known as cartesian join.

Suppose that we are sitting in a coffee shop and we decide to order breakfast. Shortly, we will look at the menu and we will start thinking of which meal and drink combination could be more tastier. Our brain will receive this signal and begin to generate all meal and drink combinations.

The following image illustrates all menu combinations that can be generated by our brain. The SQL CROSS JOIN works similarly to this mechanism, as it creates all paired combinations of the rows of the tables that will be joined.



“Please don’t worry, even if you feel a bit hungry now, you can eat whatever you want after reading our article.”

The main idea of the CROSS JOIN is that it returns the Cartesian product of the joined tables. In the following tip, we will briefly explain the Cartesian product;

**Tip:** What is the Cartesian Product?

The Cartesian Product is a multiplication operation in the set theory that generates all ordered pairs of the given sets. Suppose that, A is a set and elements are {a,b} and B is a set and elements are {1,2,3}. The Cartesian Product of these two A and B is denoted AxB and the result will be like the following.

AxB ={(a,1), (a,2), (a,3), (b,1), (b,2), (b,3)}

## Syntax

The syntax of the CROSS JOIN in SQL will look like the below syntax:

SELECT ColumnName\_1,

ColumnName\_2,

ColumnName\_N

FROM [Table\_1]

CROSS JOIN [Table\_2]

Or we can use the following syntax instead of the previous one. This syntax does not include the CROSS JOIN keyword; only we will place the tables that will be joined after the FROM clause and separated with a comma.

SELECT ColumnName\_1,

ColumnName\_2,

ColumnName\_N

FROM [Table\_1],[Table\_2]

The resultset does not change for either of these syntaxes. In addition, we must notice one point about the CROSS JOIN. Unlike the INNER JOIN, LEFT JOIN and FULL OUTER JOIN, the CROSS JOIN does not require a joining condition.

## SQL CROSS JOIN example:

In this example, we will consider the breakfast menu example again, which we mentioned in the earlier part of the article. Firstly, we will create the two-sample tables which contain the drink and meal names. After then, we will populate them with some sample data.

Through the following query, we will perform these two-steps:

1

2

3

4

5

6

7

8

9

10

CREATE TABLE Meals(MealName VARCHAR(100))

CREATE TABLE Drinks(DrinkName VARCHAR(100))

INSERT INTO Drinks

VALUES('Orange Juice'), ('Tea'), ('Cofee')

INSERT INTO Meals

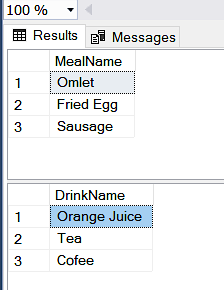
VALUES('Omlet'), ('Fried Egg'), ('Sausage')

SELECT \*

FROM Meals;

SELECT \*

FROM Drinks

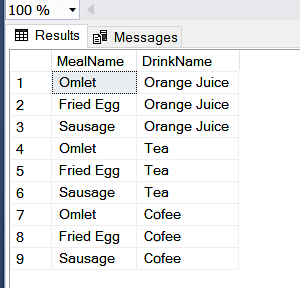


The following query will join the Meals and Drinks table with the CROSS JOINkeyword and we will obtain all of the paired combinations of the meal and drink names.

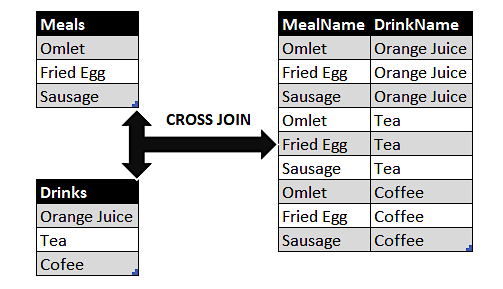
2

SELECT \* FROM Meals

CROSS JOIN Drinks



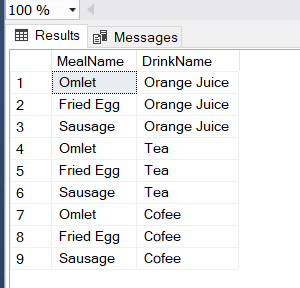
The below image illustrates the working principle of the CROSS JOIN.



At the same time, we can use the following query in order to obtain the same result set with an alternative syntax without CROSS JOIN.

SELECT \* FROM Meals

,Drinks



**Tip:** The resultset row count will equal to multiplication of tables row counts that will be joined. For the breakfast menu example, the Meals table row count is 3 and the Drinks table row count is 3, so the resultset row count can find with the following calculation.

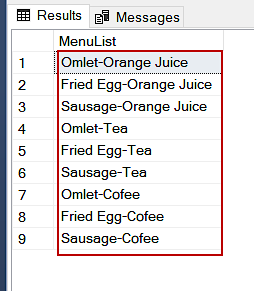
**3** (Meals table row count) x **3** (Drinks table row count) = **9** (Resultset row count)

[CONCAT\_WS](https://docs.microsoft.com/en-us/sql/t-sql/functions/concat-ws-transact-sql?view=sql-server-ver15) function will help to concatenate the column expressions. Thus, we can create a more meaningful breakfast menu resultset.

SELECT CONCAT\_WS('-',MealName,DrinkName) AS MenuList

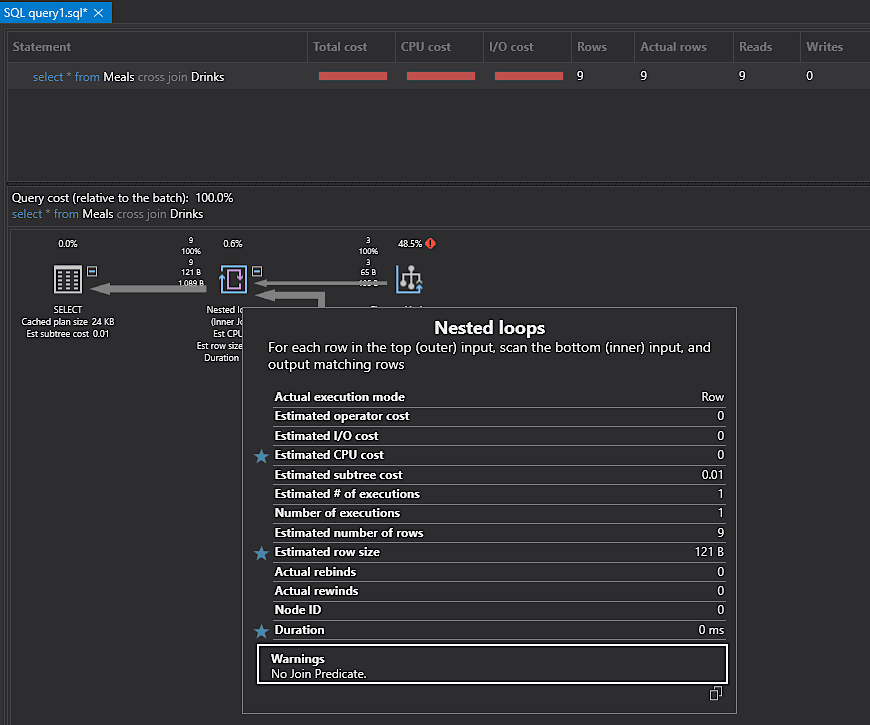
FROM Meals CROSS JOIN

Drinks



## SQL CROSS JOIN and Performance Considerations

The SQL queries which contain the CROSS JOIN keyword can be very costly. We try to say that these queries have a high potential to consume more resources and can cause performance issues. For the following query, we will analyze the execution plan with [ApexSQL Plan](https://www.apexsql.com/sql-tools-plan.aspx). In the generated actual execution plan, we will see a Nested loops operator and when we hover over the mouse on this operator, the detail pop-up window will appear.



In this window, a warning message leaps to our eyes. “No Join Predicate” message specifies that this query can be faced with performance problems. For this reason, the query optimizer warns us about this potential problem. Briefly, when we decide to use the CROSS JOIN in any query, we should consider the number of the tables that will be joined. Such as, when we CROSS JOIN two tables and if the first one contains 1000 rows and the second one contains 1000 rows, the row count of the resultset will be 1.000.000 rows.

**Tip:** CROSS JOIN can only be implemented with Nested Loops, so the following queries will return an error if we force Query Optimizer to use other join types.

SELECT \* FROM Meals

CROSS JOIN Drinks

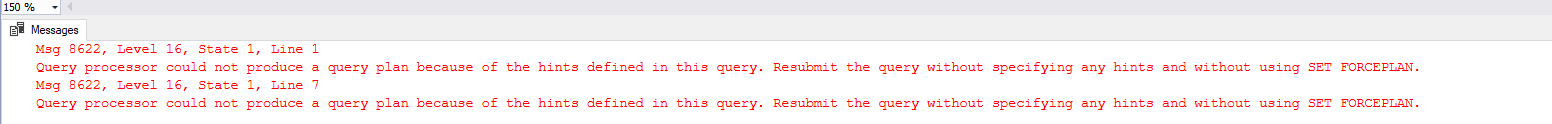
OPTION (MERGE JOIN )

GO

SELECT \* FROM Meals

CROSS JOIN Drinks

OPTION (HASH JOIN )



## Conclusion

In this article, we learned SQL CROSS JOIN basics with details and we also mentioned the performance considerations of the CROSS JOIN. When CROSS JOIN is used for tables that have a high number of rows, it might affect the performance negatively.

# UNION

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [UNION](https://dashboard.stige.in/index.php/topic/union-2/)

In Progress

The UNION operator is used to combine the result-set of two or more SELECT statements.

* Every SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in every SELECT statement must also be in the same order

### UNION Syntax

SELECT column\_name(s) FROM table1

UNION

SELECT column\_name(s) FROM table2;

### UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

SELECT column\_name(s) FROM table1

UNION ALL

SELECT column\_name(s) FROM table2;

**Note:** The column names in the result-set are usually equal to the column names in the first SELECT statement.

## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the “Customers” table:

| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the “Suppliers” table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly’s Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 |  |

## SQL UNION Example

The following SQL statement returns the cities (only distinct values) from both the “Customers” and the “Suppliers” table:

### Example

SELECT City FROM Customers

UNION

SELECT City FROM Suppliers

ORDER BY City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union)

**Note:** If some customers or suppliers have the same city, each city will only be listed once, because UNION selects only distinct values. Use UNION ALL to also select duplicate values!

## SQL UNION ALL Example

The following SQL statement returns the cities (duplicate values also) from both the “Customers” and the “Suppliers” table:

### Example

SELECT City FROM Customers

UNION ALL

SELECT City FROM Suppliers

ORDER BY City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union_all)

## SQL UNION With WHERE

The following SQL statement returns the German cities (only distinct values) from both the “Customers” and the “Suppliers” table:

### Example

SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union2)

## SQL UNION ALL With WHERE

The following SQL statement returns the German cities (duplicate values also) from both the “Customers” and the “Suppliers” table:

### Example

SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION ALL

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union_all2)

## Another UNION Example

The following SQL statement lists all customers and suppliers:

### Example

SELECT 'Customer' AS Type, ContactName, City, Country

FROM Customers

UNION

SELECT 'Supplier', ContactName, City, Country

FROM Suppliers;

[Try it yourself](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union3)

Notice the “AS Type” above – it is an alias. [SQL Aliases](https://www.w3schools.com/sql/sql_alias.asp) are used to give a table or a column a temporary name. An alias only exists for the duration of the query. So, here we have created a temporary column named “Type”, that list whether the contact person is a “Customer” or a “Supplier”.

# SET THEORY

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 4: Joins and Set Operations](https://dashboard.stige.in/index.php/lessons/task-4-joins-and-set-operations/) [SET THEORY](https://dashboard.stige.in/index.php/topic/set-theory/)

In Progress

The set theory is very important in order to understand data and databases. While you could live without it and still be a good SQL developer, understanding it will surely help a lot. So, let’s dive into the matter.

## Set Theory – Math & Logic

I guess you remember these lessons from high school. To me, this was one of the most boring parts of my education, because many things sounded so obvious and you just had new notation and operators to work with sets – again pretty obvious one. While most people won’t use that knowledge later in their life, that’s not the case for those who are into databases.

## What are Sets in the Set Theory?

Sets can contain really anything. Let’s start from the simplest possible set and that is the **empty set – S = {}**. As you can see, the empty set doesn’t contain any data. We don’t know anything about that set since it’s undefined – there are no data types or data values.

We could have a **set of numbers** **S = {1, 2, 3}**, or **T = {1, 1, 2, 3, 1001}**, or **U = {3, 1, 2}**. This is much more interesting, because we have values defined, and we can also tell that all of these sets contain numerical data.

Note. Two sets are equal if they contain same element, no matter how they are ordered. In our example, sets S = {1, 2, 3} & U = {3, 1, 2}, are the same.

Sets could also contain strings, e.g. **A = {“Jack”, “Jill”, “John”}**, **B = {“Zagreb”, “Belgrade”, “New York”, “Berlin”, “Moscow”}**.

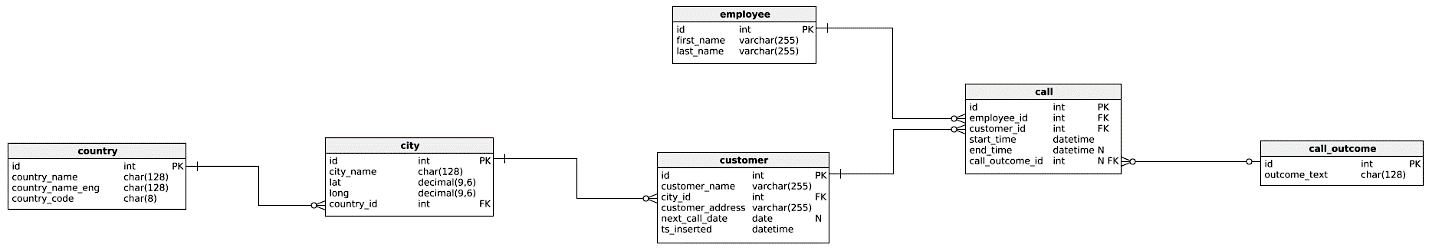
***Note:*** In the set theory, a set can contain anything, and the set elements even don’t have to be of the same type.

This is also a set: **C = {1, “Jack”, 3.14, 2020/02/14}**. It contains 4 separate information, and in this case, they have different data types.

We’re more interested in sets that contain structures/records/tuples. Let’s take a look at one such example **country = {(1, Deutschland, Germany, DEU), (2, Srbija, Serbia, SRB), (3, Hrvatska, Croatia, HRV), (4, United States of America, United States of America, USA), (5, Polska, Poland, POL), (6, España, Spain, ESP), (7, Rossiya, Russia, RUS)}**. We have a list of 7 structures containing data for 7 different countries. This is something we’ve already met, and these data are actually the contents of the **country** table from our model.

## The Model

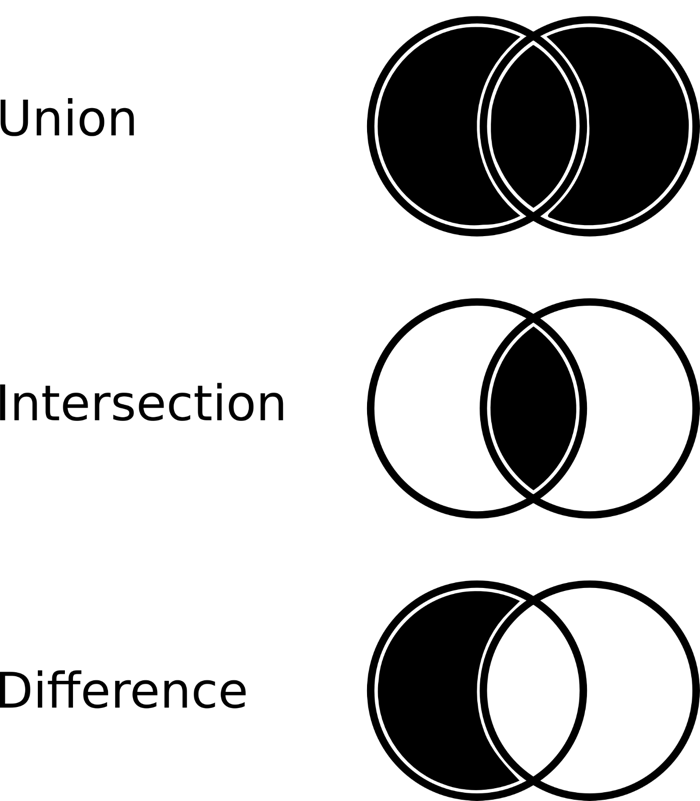
Let’s remind ourselves of the model we’re using in this article series.

[](https://www.sqlshack.com/wp-content/uploads/2020/02/the-data-model-well-use.png)

If we’re talking from the perspective of the set theory, you can look at each table as one set. Same stands for query results. Technically the result of each query is a new table and you’ll treat it in the same manner as the regular table – this query result is also a set; you can write new queries using this query as a table etc.

## Set Theory and Venn Diagrams

In SQL Server we have 3 important operators at our disposal – UNION (ALL), INTERSECT, and EXCEPT. They return the result of related operators from the set theory (on the picture below).



The easiest way to explain this is:

* UNION – Returns elements from both sets (if there are duplicates, they are in the final set, only once)
* UNION ALL – Same as the UNION operator, but will contain all duplicates
* INTERSECT – Returns a set containing elements that are present in both sets
* EXCEPT/MINUS (difference) – A MINUS B is a set containing elements from the set A that are not elements of the set B (so A MINUS (A INTERSECT B))

We won’t analyze situations where sets don’t have any common elements (A UNION B = all elements from A and B, A INTERSECT B = {}, A EXCEPT B = A, B EXCEPT A = B) and where set A = set B (A UNION B = A = B, A INTRSECT B = A = B, A EXCEPT B = B EXCEPT A = {}).

## Set Theory and SQL

We talked a lot about the set theory so far, and now it’s time for some practice. We’ll write down a few queries which will show how UNION (ALL), INTERSECT and EXCEPT operators work.

#### #1 First we’ll test two separate queries and analyze the result set they return

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

-- list all customers with exactly 3 calls

select customer.\*

from customer

where id in (

select customer.id

from customer

inner join call on customer.id = call.customer\_id

group by customer.id

having count(\*) = 3

);

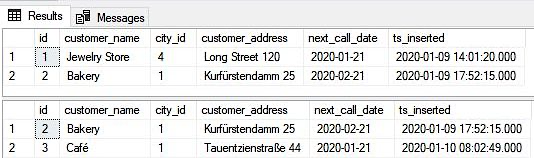
-- list all customers from Berlin

select customer.\*

from customer

inner join city on customer.city\_id = city.id

where city.city\_name = 'Berlin';



You should notice a few things:

* The first query returns all customers having exactly 3 calls
* The second query returns all customers from Berlin
* Both queries return the same columns, but rows returned are not the same. This is important because you can use operators working with sets only if these two sets are composed of elements with the same structure
* Each result set has 2 rows. “Bakery” is present in both result sets, and each set has one other row

#### #2 UNION and UNION ALL

Now we’ll use two available UNION operators. Any of these operators (UNION (ALL), INTERSECT, EXCEPT) is used in a way you just place it between queries.

-- UNION

-- list all customers with exactly 3 calls

select customer.\*

from customer

where id in (

select customer.id

from customer

inner join call on customer.id = call.customer\_id

group by customer.id

having count(\*) = 3

)

UNION

-- list all customers from Berlin

select customer.\*

from customer

inner join city on customer.city\_id = city.id

where city.city\_name = 'Berlin';

-- UNION ALL

-- list all customers with exactly 3 calls

select customer.\*

from customer

where id in (

select customer.id

from customer

inner join call on customer.id = call.customer\_id

group by customer.id

having count(\*) = 3

)

UNION ALL

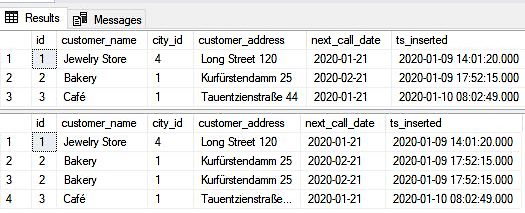
-- list all customers from Berlin

select customer.\*

from customer

inner join city on customer.city\_id = city.id

where city.city\_name = 'Berlin';



For these two result sets returned, you should notice the following:

* Each query, the first one using UNION, and the second one using UNION ALL returns 1 result set
* The result set returned by the UNION query returned all rows returned by the two queries used. The only difference is that the duplicated row had been eliminated
* The query using UNION ALL returned all rows from both queries, without removing duplicates
* The UNION is used more often, and you’ll probably use it when you have a few complex queries and you simply want to “join” their results without writing a single more complex query

### #3 INTERSECT

The INTERSECT should return elements/rows which appear in both sets.

-- INTERSECT

-- list all customers with exactly 3 calls

select customer.\*

from customer

where id in (

select customer.id

from customer

inner join call on customer.id = call.customer\_id

group by customer.id

having count(\*) = 3

)

INTERSECT

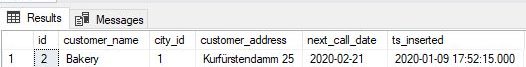
-- list all customers from Berlin

select customer.\*

from customer

inner join city on customer.city\_id = city.id

where city.city\_name = 'Berlin';



Everything went as expected and you can see that the “Bakery” row was returned as a result.

### #4 EXCEPT

The EXCEPT operator returns all elements/rows from the first set, except those that are in the second set.

-- A EXCEPT B

-- list all customers with exactly 3 calls

select customer.\*

from customer

where id in (

select customer.id

from customer

inner join call on customer.id = call.customer\_id

group by customer.id

having count(\*) = 3

)

EXCEPT

-- list all customers from Berlin

select customer.\*

from customer

inner join city on customer.city\_id = city.id

where city.city\_name = 'Berlin';

-- B EXCEPT A

-- list all customers from Berlin

select customer.\*

from customer

inner join city on customer.city\_id = city.id

where city.city\_name = 'Berlin'

EXCEPT

-- list all customers with exactly 3 calls

select customer.\*

from customer

where id in (

select customer.id

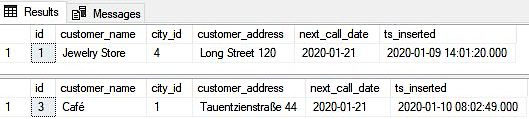
from customer

inner join call on customer.id = call.customer\_id

group by customer.id

having count(\*) = 3

);



The most important thing you should notice here is:

* A EXCEPT B is not the same as B EXCEPT A (A and B are names of the sets)
* The first query returns all customers having exactly 3 calls except those from Berlin, while the second query finds and returns customers from Berlin except those with exactly 3 calls

## Why Should You Understand the Set Theory?

Theory in IT is not so “hot” as the practice is. Same stands for the set theory. Still, understanding what lies in the background of the operations you run is essential for better understanding not only these operators but databases themselves.

# Task 5: Nested Queries, Views, CTEs

Video - <https://youtu.be/9pC0k6vGyK0>

# Nested Queries in SQL

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Nested Queries, Views, CTEs](https://dashboard.stige.in/index.php/lessons/task-5-nested-queries-views-ctes/) [Nested Queries in SQL](https://dashboard.stige.in/index.php/topic/nested-queries-in-sql/)

In Progress

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use **STUDENT, COURSE, STUDENT\_COURSE** tables for understanding nested queries.

### **STUDENT**

| **S\_ID** | **S\_NAME** | **S\_ADDRESS** | **S\_PHONE** | **S\_AGE** |
| --- | --- | --- | --- | --- |
| S1 | RAM | DELHI | 9455123451 | 18 |
| S2 | RAMESH | GURGAON | 9652431543 | 18 |
| S3 | SUJIT | ROHTAK | 9156253131 | 20 |
| S4 | SURESH | DELHI | 9156768971 | 18 |

### **COURSE**

| **C\_ID** | **C\_NAME** |
| --- | --- |
| C1 | DSA |
| C2 | Programming |
| C3 | DBMS |

### **STUDENT\_COURSE**

| **S\_ID** | **C\_ID** |
| --- | --- |
| S1 | C1 |
| S1 | C3 |
| S2 | C1 |
| S3 | C2 |
| S4 | C2 |
| S4 | C3 |

# Types of Nested Queries

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Nested Queries, Views, CTEs](https://dashboard.stige.in/index.php/lessons/task-5-nested-queries-views-ctes/) [Types of Nested Queries](https://dashboard.stige.in/index.php/topic/types-of-nested-queries/)

In Progress

There are mainly two types of nested queries:

### **Independent Nested Queries**

In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

**IN:** If we want to find out S\_ID who are enrolled in C\_NAME ‘DSA’ or ‘DBMS’, we can write it with the help of independent nested query and IN operator. From COURSE table, we can find out C\_IDfor C\_NAME ‘DSA’ or DBMS’ and we can use these C\_IDs for finding S\_IDs from STUDENT\_COURSE TABLE.

**STEP 1:** Finding **C\_ID** for **C\_NAME** =’DSA’ or ‘DBMS’

Select C\_ID from COURSE where C\_NAME = ‘DSA’ or C\_NAME = ‘DBMS’

**STEP 2:** Using **C\_ID** of step 1 for finding **S\_ID**

Select S\_ID from STUDENT\_COURSE where C\_ID IN

(SELECT C\_ID from COURSE where C\_NAME = ‘DSA’ or C\_NAME=’DBMS’);

The inner query will return a set with members C1 and C3 and outer query will return those **S\_ID**s for which **C\_ID** is equal to any member of set (C1 and C3 in this case). So, it will return S1, S2 and S4.

**Note:** If we want to find out names of STUDENTs who have either enrolled in ‘DSA’ or ‘DBMS’, it can be done as:

Select S\_NAME from STUDENT where S\_ID IN

(Select S\_ID from STUDENT\_COURSE where C\_ID IN

(SELECT C\_ID from COURSE where C\_NAME=’DSA’ or C\_NAME=’DBMS’));

**NOT IN:**If we want to find out S\_IDs of STUDENTs who have neither enrolled in ‘DSA’ nor in ‘DBMS’, it can be done as:

Select S\_ID from STUDENT where S\_ID NOT IN

(Select S\_ID from STUDENT\_COURSE where C\_ID IN

(SELECT C\_ID from COURSE where C\_NAME=’DSA’ or C\_NAME=’DBMS’));

The innermost query will return a set with members C1 and C3. Second inner query will return those **S\_ID**s for which **C\_ID** is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those **S\_ID**s where **S\_ID** is not a member of set (S1, S2 and S4). So it will return S3.

### **Co-related Nested Queries**

 In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. e.g.; If we want to find out S\_NAME of STUDENTs who are enrolled in C\_ID ‘C1’, it can be done with the help of co-related nested query as:

Select S\_NAME from STUDENT S where EXISTS

( select \* from STUDENT\_COURSE SC where S.S\_ID=SC.S\_ID and SC.C\_ID=’C1’);

For each row of **STUDENT** S, it will find the rows from **STUDENT\_COURSE** where S.**S\_ID** = SC.**S\_ID** and SC.**C\_ID**=’C1’. If for a **S\_ID** from **STUDENT** S, atleast a row exists in **STUDENT\_COURSE** SC with **C\_ID**=’C1’, then inner query will return true and corresponding **S\_ID** will be returned as output.

# SQL | Views

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Nested Queries, Views, CTEs](https://dashboard.stige.in/index.php/lessons/task-5-nested-queries-views-ctes/) [SQL | Views](https://dashboard.stige.in/index.php/topic/sql-views/)

In Progress

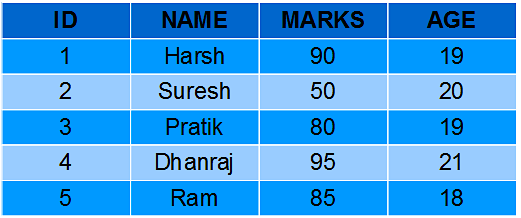
Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition.

In this article we will learn about creating , deleting and updating Views.  
**Sample Tables**:

#### StudentDetails



#### StudentMarks



# Creating Views

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Nested Queries, Views, CTEs](https://dashboard.stige.in/index.php/lessons/task-5-nested-queries-views-ctes/) [Creating Views](https://dashboard.stige.in/index.php/topic/creating-views/)

In Progress

We can create View using **CREATE VIEW** statement. A View can be created from a single table or multiple tables.

**Syntax**:

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE condition;

view\_name: Name for the View

table\_name: Name of the table

condition: Condition to select rows

### **Examples**:

#### **Creating View from a single table:**

* In this example we will create a View named DetailsView from the table StudentDetails.  
  Query:

CREATE VIEW DetailsView AS

SELECT NAME, ADDRESS

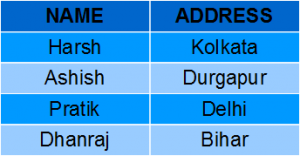
FROM StudentDetails

WHERE S\_ID < 5;

To see the data in the View, we can query the view in the same manner as we query a table.

SELECT \* FROM DetailsView;

Output:



* In this example, we will create a view named StudentNames from the table StudentDetails.

Query:

CREATE VIEW StudentNames AS

SELECT S\_ID, NAME

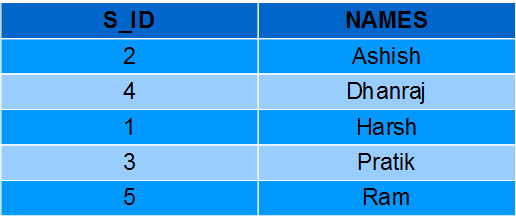
FROM StudentDetails

ORDER BY NAME;

If we now query the view as,

SELECT \* FROM StudentNames;

Output:



#### **Creating View from multiple tables**:

In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks. To create a View from multiple tables we can simply include multiple tables in the SELECT statement.

Query:

CREATE VIEW MarksView AS

SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS

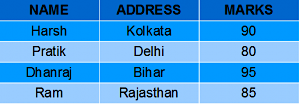
FROM StudentDetails, StudentMarks

WHERE StudentDetails.NAME = StudentMarks.NAME;

To display data of View MarksView:

SELECT \* FROM MarksView;

Output:



# Deleting Views

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Nested Queries, Views, CTEs](https://dashboard.stige.in/index.php/lessons/task-5-nested-queries-views-ctes/) [Deleting Views](https://dashboard.stige.in/index.php/topic/deleting-views/)

In Progress

We have learned about creating a View, but what if a created View is not needed any more? Obviously we will want to delete it. SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

**Syntax**:

DROP VIEW view\_name;

**view\_name**: Name of the View which we want to delete.

For example, if we want to delete the View **MarksView**, we can do this as:

DROP VIEW MarksView;

# Updating Views

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Nested Queries, Views, CTEs](https://dashboard.stige.in/index.php/lessons/task-5-nested-queries-views-ctes/) [Updating Views](https://dashboard.stige.in/index.php/topic/updating-views/)

In Progress

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

* We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.  
  **Syntax**:

CREATE OR REPLACE VIEW view\_name AS

SELECT column1,coulmn2,..

FROM table\_name

WHERE condition;

For example, if we want to update the view **MarksView** and add the field AGE to this View from **StudentMarks**Table, we can do this as:

CREATE OR REPLACE VIEW MarksView AS

SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS, StudentMarks.AGE

FROM StudentDetails, StudentMarks

WHERE StudentDetails.NAME = StudentMarks.NAME;

If we fetch all the data from MarksView now as:

SELECT \* FROM MarksView;

Output:



* **Inserting a row in a view**:

We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.

**Syntax**:

INSERT INTO view\_name(column1, column2 , column3,..)

VALUES(value1, value2, value3..);

view\_name: Name of the View

**Example**:  
In the below example we will insert a new row in the View DetailsView which we have created above in the example of “creating views from a single table”.

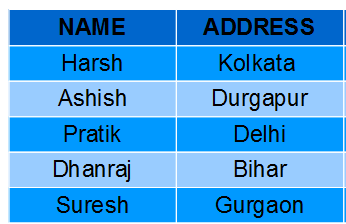
INSERT INTO DetailsView(NAME, ADDRESS)

VALUES("Suresh","Gurgaon");

If we fetch all the data from DetailsView now as,

SELECT \* FROM DetailsView;

Output:



* **Deleting a row from a View**:

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

**Syntax**:

DELETE FROM view\_name

WHERE condition;

view\_name:Name of view from where we want to delete rows

condition: Condition to select rows

**Example**:  
In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

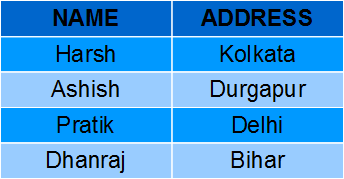
DELETE FROM DetailsView

WHERE NAME="Suresh";

If we fetch all the data from DetailsView now as,

SELECT \* FROM DetailsView;

Output:



# With Check Option

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Nested Queries, Views, CTEs](https://dashboard.stige.in/index.php/lessons/task-5-nested-queries-views-ctes/) [With Check Option](https://dashboard.stige.in/index.php/topic/with-check-option/)

In Progress

The WITH CHECK OPTION clause in SQL is a very useful clause for views. It is applicable to a updatable view. If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.

* The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.
* If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.

**Example**:  
In the below example we are creating a View SampleView from StudentDetails Table with WITH CHECK OPTION clause.

CREATE VIEW SampleView AS

SELECT S\_ID, NAME

FROM StudentDetails

WHERE NAME IS NOT NULL

WITH CHECK OPTION;

In this View if we now try to insert a new row with null value in the NAME column then it will give an error because the view is created with the condition for NAME column as NOT NULL.  
For example,though the View is updatable but then also the below query for this View is not valid:

INSERT INTO SampleView(S\_ID)

VALUES(6);

**NOTE**: The default value of NAME column is null.

# Uses of a View

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 5: Nested Queries, Views, CTEs](https://dashboard.stige.in/index.php/lessons/task-5-nested-queries-views-ctes/) [Uses of a View](https://dashboard.stige.in/index.php/topic/uses-of-a-view/)

In Progress

A good database should contain views due to the given reasons:

1. **Restricting data access –**  
   Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
2. **Hiding data complexity –**  
   A view can hide the complexity that exists in a multiple table join.
3. **Simplify commands for the user –**  
   Views allows the user to select information from multiple tables without requiring the users to actually know how to perform a join.
4. **Store complex queries –**  
   Views can be used to store complex queries.
5. **Rename Columns –**  
   Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement. Thus, renaming helps to to hide the names of the columns of the base tables.
6. **Multiple view facility –**  
   Different views can be created on the same table for different users.

# Task 6: Advance SQL

# What is Regex?

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [What is Regex?](https://dashboard.stige.in/index.php/topic/what-is-regex/)

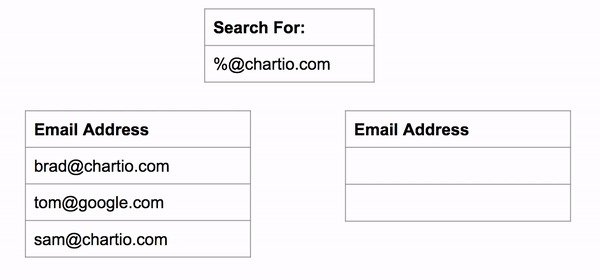
In Progress

Regex, or Regular Expressions, is a sequence of characters, used to search and locate specific sequences of characters that match a pattern.

In SQL if you were looking for email addresses from the same company Regex lets you define a pattern using comparators and [Metacharacters](https://www.ibm.com/support/knowledgecenter/en/SSGU8G_12.1.0/com.ibm.dbext.doc/ids_dbxt_545.htm), in this case using ~\* and % to help define the pattern:

SELECT \* FROM Email Addresses

WHERE Email Address ~\* '%@chartio.com'



# Using Regex in PostgreSQL

### Metacharacters

Here is a quick cheat sheet for metacharacters to help define the pattern:

| **METACHARACTER** | **DESCRIPTION** | **EXAMPLE** | **EXAMPLE MATCHES** |
| --- | --- | --- | --- |
| ^ | Start the match at the beginning of a string | ^c% | cat, car, chain |
| | | Alternation (either of two alternatives) | c(a|o)% | can, corn, cop |
| () | Group items in a single logical item | c(a|o)% | can, corn, cop |
| \_ | Any single character (using LIKE and SIMILAR TO) | c\_ | co, fico, pico |
| % | Any string (using LIKE and SIMILAR TO) | c% | chart, articulation, crate |
| . | Any single character (using POSIX) | c. | co, fico, pico |
| .\* | Any string (using POSIX) | c.\* | chart, articulation, crate |
| + | Repetition of the previous item one or more times | co+ | coo, cool |

### Comparators

There are three ways to use regex comparisons in SQL:

* LIKE
* SIMILAR TO
* POSIX comparators

LIKE and SIMILAR TO are used for basic comparisons where you are looking for a matching string. LIKE and SIMILAR TO both look and compare string patterns, the only difference is that SIMILAR TO uses the SQL99 definition for regular expressions and LIKE uses PSQL’s definition for regular expressions.

**Syntax**: [String or Column name] LIKE/SIMILAR TO [Regex]

| **EXPRESSION** | **RETURNS** |
| --- | --- |
| ‘char’ LIKE ‘char’ | True |
| ‘char’ LIKE ‘c%’ | True |
| ‘char’ LIKE ‘ha’ | True |
| ‘char’ LIKE ‘c’ | False |
| ‘char’ SIMILAR TO ‘char’ | True |
| ‘char’ SIMILAR TO ‘%(h|g)%’ | True |
| ‘char’ SIMILAR TO ‘h’ | False |
| ‘char’ SIMILAR TO ‘(a|b)%’ | False |

Unlike LIKE and SIMILAR TO, POSIX is not a keyword that is used in a SQL query. POSIX is a set of comparators for case matches and non equivalency. It is the most powerful way to use Regex in SQL. Regex does not use = and != to compare rather it uses these POSIX comparators:

1. ~ : Case-sensitive, compares two statements, returns true if the first string is contained in the second
2. ~\* : Case-insensitive, compares two statements, returns true if the first string is contained in the second
3. !~ : Case-sensitive, compares two statements, returns false if the first string is contained in the second
4. !~\* : Case-insensitive, compares two statements, return false if the first string is contained in the second

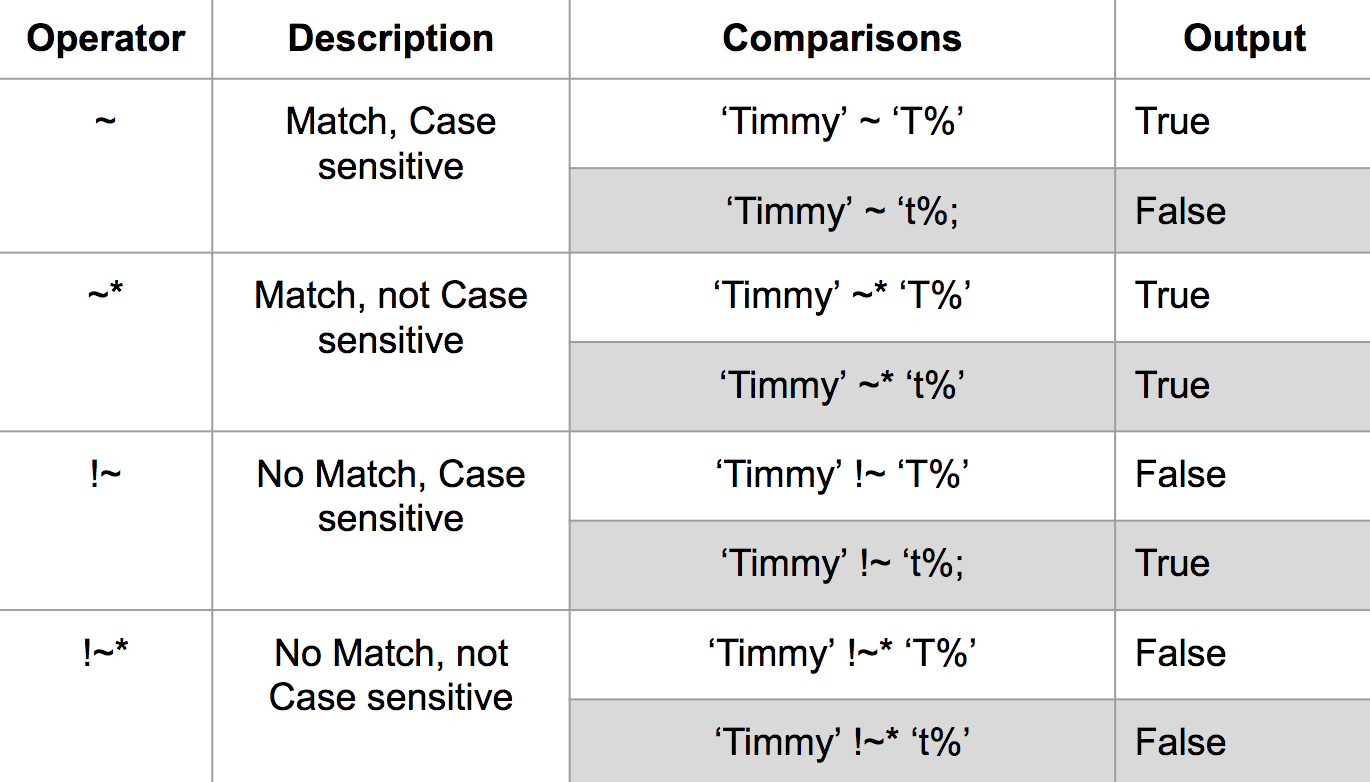
**Syntax**: [String or Column name] [POSIX] [Regex]

These comparators can be used in queries to locate or exclude certain data from being returned.

# Examples of Regex in SQL Queries

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Examples of Regex in SQL Queries](https://dashboard.stige.in/index.php/topic/examples-of-regex-in-sql-queries/)

In Progress

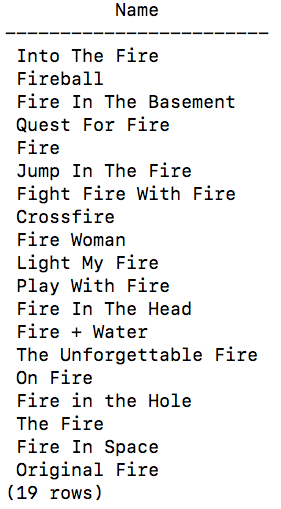


If you wanted to search a column of a database for all entries that contain the word ‘fire’, you could use ~\* ‘fire’  to find any row that contains the word:

SELECT (column name)

FROM (table name)

WHERE (column name) ~\* 'fire';

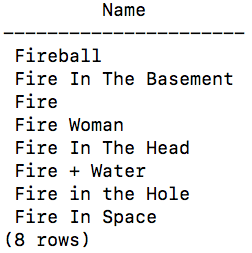


To get all entries that start with the word ‘Fire’:

SELECT (column name)

FROM (table name)

WHERE (column name) ~ \* '^fire';



A full list of regular expressions can be found at: [RexEgg](https://www.rexegg.com/regex-quickstart.html)

# Regex Summary

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Regex Summary](https://dashboard.stige.in/index.php/topic/regex-summary/)

In Progress

* Regular expressions use patterns to match strings.
* Regex provides a way to query databases to find a smaller subset of data.
* The POSIX comparators are:
* ~ : Case-sensitive, compares two statements, returns true if the first is contained in the second
* ~\* : Case-insensitive, compares two statements, returns true if the first is contained in the second
* !~ : Case-sensitive, compares two statements, returns false if the first is contained in the second
* !~\* : Case-insensitive, compares two statements, return false if the first is contained in the second

# Window Function

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Window Function](https://dashboard.stige.in/index.php/topic/window-function/)

In Progress

Data is proliferating at an astonishing rate – growing to 44 zettabytes in 2020! And it goes without saying that the technology to handle such mammoth amounts of data is also changing at a commensurate rate.

Today we have a panoply of tools like [Hive](https://www.analyticsvidhya.com/blog/2020/10/getting-started-with-apache-hive/?utm_source=blog&utm_medium=window-function-a-must-know-sql-concept) and [Spark](https://www.analyticsvidhya.com/blog/2019/10/pyspark-for-beginners-first-steps-big-data-analysis/?utm_source=blog&utm_medium=window-function-a-must-know-sql-concept) to handle [Big Data](https://www.analyticsvidhya.com/blog/2020/11/what-is-big-data-a-quick-introduction-for-analytics-and-data-engineering-beginners/). But, even though they differ in certain aspects, they still employ the fundamentals of SQL, making it very easy for people from all walks of life to manipulate Big Data with a breeze. Although we still falter at certain aspects of SQL. So in this article, I am going to talk about one such aspect in particular – Window Functions.



That’s right! There is such a thing as Window Functions in SQL, I kid you not! And going by the flabbergasted look on your face, this article seems like the need of the hour. Even I wasn’t too aware of these functions until recently which goes to show how underappreciated these functions are.

But wait till the end of this article because Window Functions will really blow your mind away with the simplicity with which they solve such complex problems. And yes, [Data Engineers](https://www.analyticsvidhya.com/blog/2018/11/data-engineer-comprehensive-list-resources-get-started/?utm_source=blog&utm_medium=window-function-a-must-know-sql-concept), Data Scientists, Data Analysts, and everyone else out there flirting with data needs to give these functions their due credit!

Before going forward, I suggest you get comfortable with the basic SQL functions by going over this article – [*24 commonly used SQL functions*](https://www.analyticsvidhya.com/blog/2020/07/sql-functions-for-data-analysis-tasks/?utm_source=blog&utm_medium=window-function-a-must-know-sql-concept). And if you are interested in learning SQL in a course format, please refer to our course – [*Structured Query Language (SQL) for Data Science*](https://courses.analyticsvidhya.com/courses/structured-query-language-sql-for-data-science?utm_source=blog&utm_medium=window-function-a-must-know-sql-concept).

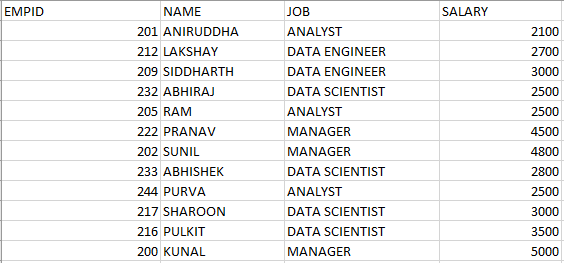
Video - <https://youtu.be/TzsrO4zTQj8>

# Introducing the Dataset

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Introducing the Dataset](https://dashboard.stige.in/index.php/topic/dataset/)

In Progress

Before going further, let me introduce you to the dummy dataset we will be working on in this article. Assume there is a company that maintains the employee’s name, job, and salary records as follows:



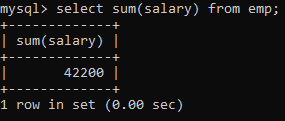
We are going to use this sample dataset to understand the concepts in this article. Alright, let’s get started!

# Where do Aggregate function lag?

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Where do Aggregate function lag?](https://dashboard.stige.in/index.php/topic/where-do-aggregate-function-lag/)

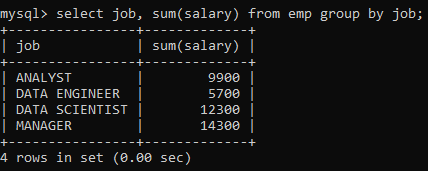
In Progress

Suppose you want to determine the total salary of all the employees in the company. How would you go about it? You can simply use the SUM() aggregate function on the SALARY column.



Easy.

How about determining the total salary of employees per job category? Use the last query and append a GROUP BY clause on the JOB column.



Great!

Now let me pose two more questions for you:

1. Display the total salary and the total salary per job category along with every row value.
2. Arrange the salary in a decreasing order within each job category.



Did you get it?  No? Probably?

These definitely weren’t as easy as the first ones that you were able to get instantly. But why?

Well, if you think about it, the former queries required simple aggregate functions to solve the problem. SQL aggregate functions only give us a single value for the group of rows aggregated together (think of the first query we wrote).

But the latter queries couldn’t simply be solved using such functions. Those queries want us to maintain the original identity of the individual rows, something that the aggregate functions fail to address. Therefore, to solve such queries we need different kinds of functions – the Window functions.

# What are Window Functions in SQL ?

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [What are Window Functions in SQL ?](https://dashboard.stige.in/index.php/topic/what-are-window-functions-in-sql/)

In Progress

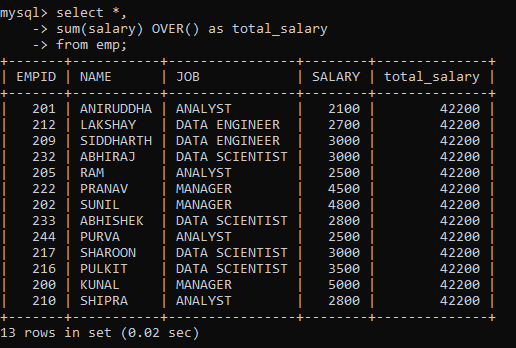
**Window functions** perform calculations on a set of rows that are related together. But, unlike the aggregate functions, windowing functions do not collapse the result of the rows into a single value. Instead, all the rows maintain their original identity and the calculated result is returned for every row.

# Understanding SQL Window Functions – Over Clause

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Understanding SQL Window Functions – Over Clause](https://dashboard.stige.in/index.php/topic/understanding-sql-window-functions-over-clause/)

In Progress

For example, if I were to display the total salary of employees along with every row value, it would look something like this:



The **OVER** clause signifies a window of rows over which a window function is applied. It can be used with aggregate functions, like we have used with the SUM function here, thereby turning it into a window function. Or, it can also be used with non-aggregate functions that are only used as window functions (we will learn more about them in the later sections).

So the syntax for defining a simple window function that outputs the same value for all rows is as follows:

window\_function\_name(<expression>) OVER ( )

But, how about applying the window function to specific rows instead of on the entire table?

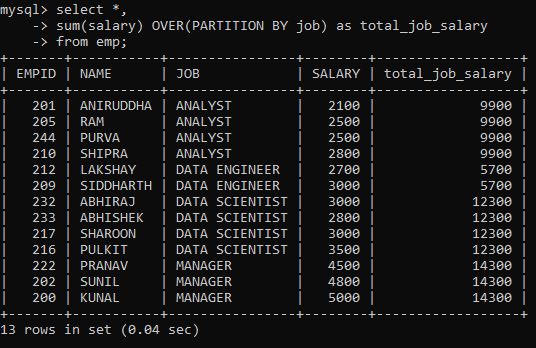
# Windowing with PARTITION BY

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Windowing with PARTITION BY](https://dashboard.stige.in/index.php/topic/windowing-with-partition-by/)

In Progress

The **PARTITION BY** clause is used in conjunction with the OVER clause. It breaks up the rows into different partitions. These partitions are then acted upon by the window function.

For example, to display the total salary per job category for all the rows we would have to modify our original SQL query as follows:



As you can see, the total\_job\_salary column depicts the sum of sales for that specific job category and not for the entire table.

So, the syntax for defining window function for the partition of rows is as follows:

window\_function\_name(<expression>) OVER (<partition\_by\_clause>)

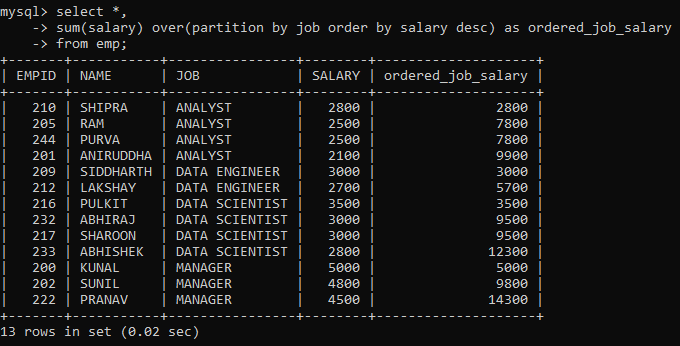
Now, how about arranging the rows within each partition?

# Arranging Rows with Partitions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Arranging Rows with Partitions](https://dashboard.stige.in/index.php/topic/arranging-rows-with-partitions/)

In Progress

We know that to arrange rows in a table, we can use the ORDER BY clause. So, to arrange rows within each partition, we have to modify the OVER clause with the ORDER BY clause.



Here, the rows have been partitioned as per their job category as indicated by the JOB column. As you scroll down, you will notice the SALARY column has been ordered in descending order and the ordered\_job\_salary column depicts the running total of the job category (starting over after every partition).

So, the syntax for defining window function for the partition of rows and arranging them in order is as follows:

window\_function\_name(<expression>) OVER (<partition\_by\_clause> <order\_clause>)

# Window Functions

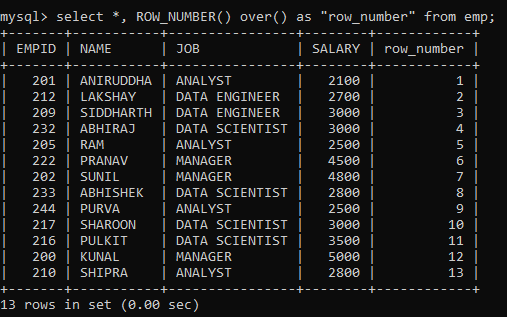
[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [Window Functions](https://dashboard.stige.in/index.php/topic/window-functions/)

In Progress

Now that we know how to define window functions using the OVER clause and some of its modified versions, we can finally move on to working with the window functions!

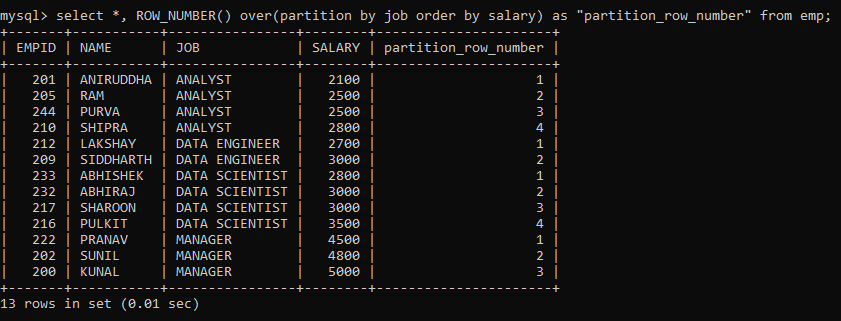
### 1. Row\_Number

Sometimes your dataset might not have a column depicting the sequential order of the rows, as is the case with our dataset. In that case, we can make use of the **ROW\_NUMBER()** window function. It assigns a unique sequential number to each row of the table.



Notice that the numbering starts from 1. Also, to prevent any clash with the MySQL keyword for the function, I have put the column name within quotes.

But, since it is a window function, we can also limit it to partitions and then order those partitions.

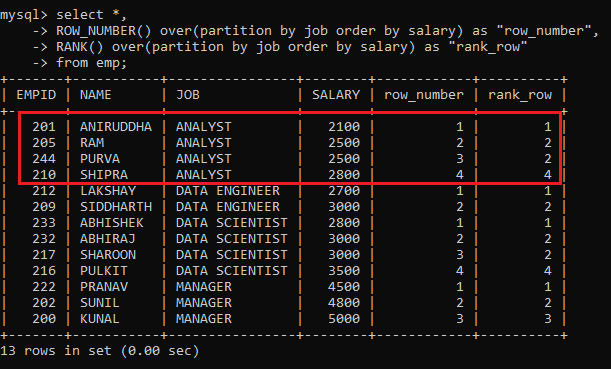


Here, we have partitioned the rows on the JOB column and ordered them based on the SALARY of the employee. Notice how the numbering restarts each time a new partition begins.

But suppose we want to rank the employees based on their salaries?

### 2. Rank vs Dense\_Rank

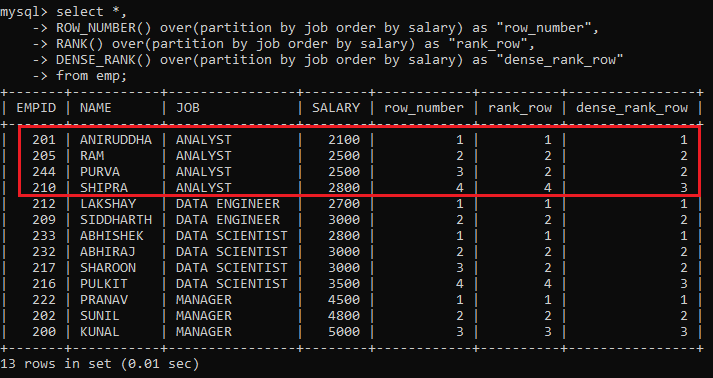
The **RANK()** window function, as the name suggests, ranks the rows within their partition based on the given condition.



Notice the highlighted portion. In the case of ROW\_NUMBER(), we have a sequential number. On the other hand, in the case of RANK(), we have the same rank for rows with the same value.

But there is a problem here. Although rows with the same value are assigned the same rank, the subsequent rank skips the missing rank. This wouldn’t give us the desired results if we had to return “top N distinct” values from a table. Therefore we have a different function to resolve this issue.

The **DENSE\_RANK()** function is similar to the RANK() except for one difference, it doesn’t skip any ranks when ranking rows.

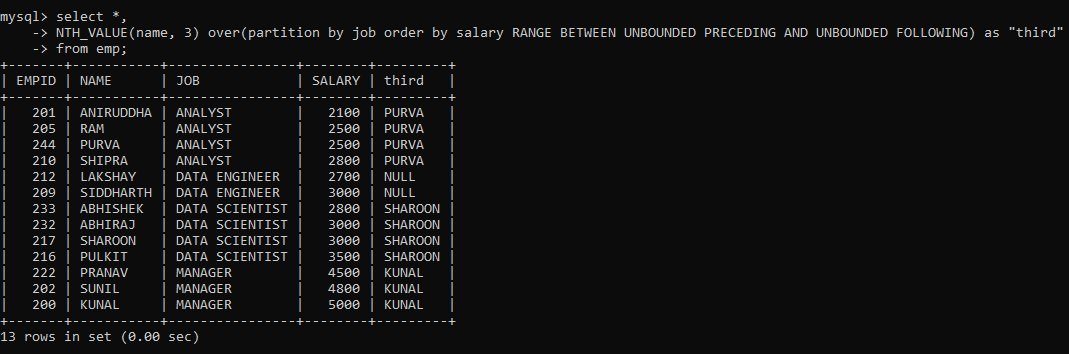


Here, all the ranks are distinct and sequentially increasing within each partition. As compared to the RANK() function, it has not skipped any rank within a partition.

### 3. Nth\_Value

If you want to retrieve the nth value from a window frame for an expression, then you can use the NTH\_VALUE(expression, N) window function.

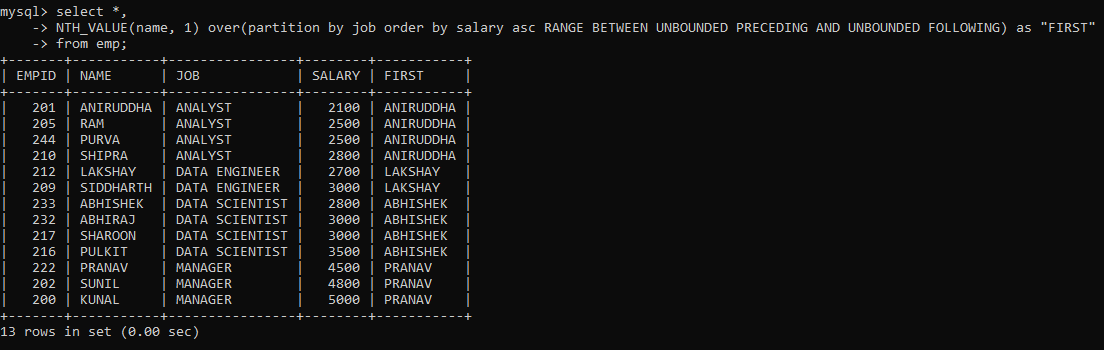
For example, to retrieve the third-highest salary in each JOB category, we can partition the rows according to the JOB column, then order the rows within the partitions according to decreasing salary, and finally, use the NTH\_VALUE function to retrieve the value. The command will be as follows:



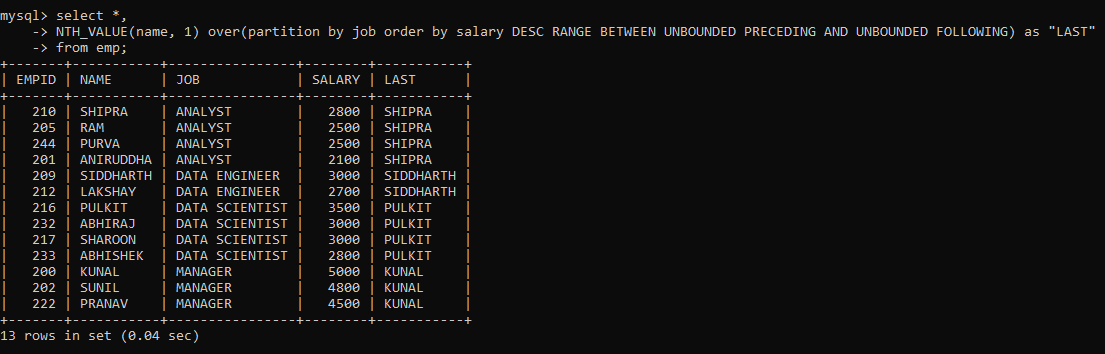
You must have noticed something different after the Order By clause. That is the [Frame clause](https://dev.mysql.com/doc/refman/8.0/en/window-functions-frames.html). It determines the subset of the partition (or frame) that will be used by the window function to calculate the value for the current row.

Here, I mentioned that all preceding and following rows for a current row be considered as within the frame when applying the window function. But why did I use the frame clause here and not with other functions? This is because the other window functions work on the entire partition even if a frame clause is provided. But only NTH\_VALUE() can work on frames within a partition.

Now suppose you wanted to output the first value from each partition? Although there is a  [FIRST\_VALUE()](https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html#function_first-value) function as well, I am going to use the NTH\_VALUE for the same.



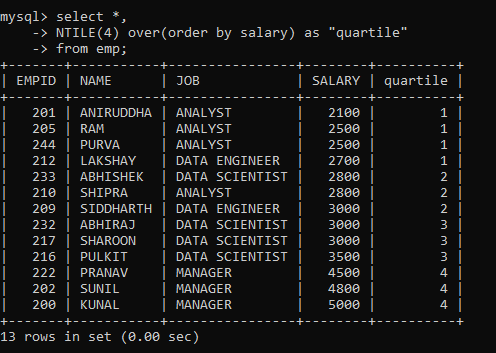
Similarly, just we also have a [LAST\_VALUE()](https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html#function_last-value) function. But I am going to determine the last value within each partition just as above, albeit using the decreasing order of rows.



### 4. Ntile

Sometimes, you might want to sort the rows within the partition into a certain number of groups. This is useful when you want to determine the percentile, quartile, etc. a particular row falls into. The**NTILE()**function is used for such purposes. It returns the group number for each of the rows in the partition.

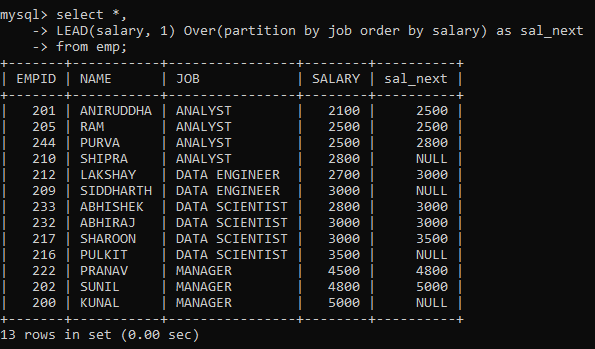
For example, let’s find the quartile for each row based on the SALARY of the employee:



Similarly, you can divide the rows into different numbers of groups and calculate the NTILE for different partitions.

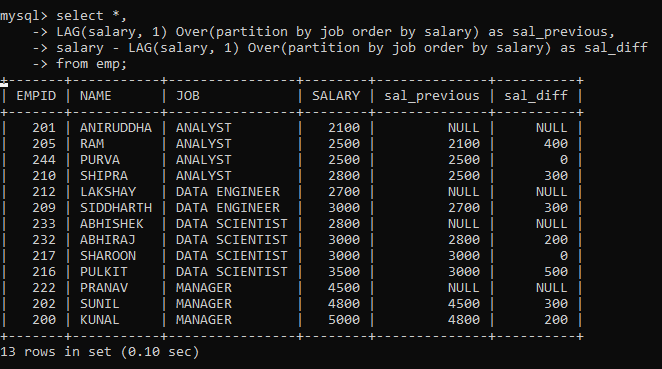
### 5. Lead and Lag

Often, you might want to compare the value of the current row to that of the preceding or succeeding row. It helps in the easy analysis of the data. The **LEAD()** and **LAG()** window functions are there just for this purpose.



Here, we created a new column containing SALARY from the next row within each partition ordered by salary using the LEAD function. Notice that the last row from each partition contains a null value because there is no succeeding row for it to pull data from.

Now, let’s do the same with the LAG function.



Here, we created two new columns. The first column contains SALARY from the previous row within each partition ordered by salary. While the second column contains the difference between SALARY from the previous row and the current row. As you can see, this is very helpful for a quick analysis of the difference between salaries within the same partition.

# SQL Code file

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 6: Advance SQL](https://dashboard.stige.in/index.php/lessons/task-6-advance-sql/) [SQL Code file](https://dashboard.stige.in/index.php/topic/sql-code-file/)

In Progress

All the SQL code related to Window Functions for this article can be found [at this link](https://gist.github.com/aniruddha27/90d2c5c5986b2966ee6deb449015747a).

# Task 7: Practice Questions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Practice Questions](https://dashboard.stige.in/index.php/lessons/task-7-practice-questions/)

In Progress

The Questions given below are for you to practice your knowledge on SQL gained till now . Try to solve the problems yourself first and then look for the solution. We recommend you solve these questions own your own as these are the typical real life examples which you will encounter while working as a data analyst.

# Joins

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Practice Questions](https://dashboard.stige.in/index.php/lessons/task-7-practice-questions/) [Joins](https://dashboard.stige.in/index.php/topic/joins-2/)

In Progress

Try the Questions given below to test your knowledge on SQL JOINS.

## **INNER JOIN**

Question : Write a query to get the cost of all transactions by user ordered by total cost descending.

#### **Input Table:**

#### Transactions table

|  |  |
| --- | --- |
| **Column** | **Type** |
| id | integer |
| user\_id | integer |
| created\_at | datetime |
| product\_id | integer |
| quantity | integer |

#### Products table

|  |  |
| --- | --- |
| **Column** | **Type** |
| id | integer |
| name | string |
| price | float |

#### Users table

|  |  |
| --- | --- |
| **Column** | **Type** |
| id | integer |
| name | string |
| created\_at | datetime |

#### **Output Table**:

|  |  |
| --- | --- |
| **Column** | **Type** |
| name | string |
| user\_id | integer |
| Total\_cost | float |

### **Possible Answer –**

SELECT  
 u.name  
 ,u.id AS user\_id  
 ,ROUND(SUM(p.price \* t.quantity ) ,2) AS total\_cost  
FROM users u  
INNER JOIN transactions t  
    ON u.id = t.user\_id  
INNER JOIN products p  
    ON p.id = t.product\_id  
GROUP BY u.name

## **LEFT JOIN**

Question : Write a query to identify the manager with the biggest team size. You may assume there is only one manager with the largest team size.

#### **Input Table**

#### Employees table

|  |  |
| --- | --- |
| **Column** | **Type** |
| id | integer |
| first\_name | string |
| last\_name | string |
| salary | integer |
| department\_id | integer |
| manager\_id | integer |

#### Managers table

|  |  |
| --- | --- |
| **Column** | **Type** |
| id | integer |
| name | string |
| team | string |

#### **Output Table:**

|  |  |
| --- | --- |
| **Column** | **Type** |
| manager | string |
| team\_size | integer |

### **Possible Answer** –

SELECT  
  m.name AS manager,  
  COUNT(e.id) AS team\_size  
FROM managers m  
LEFT JOIN employees e  
      ON e.manager\_id = m.id  
GROUP BY m.id  
ORDER BY COUNT(e.id) DESC  
LIMIT 1

## **CROSS JOIN**

Question – Say we have a table state\_streams where each row is a state and the total number of hours of streaming from a video hosting service. Write a query to get the pairs of states with total streaming amounts within 1000 of each other. For the snippet above, we would want to see something like.

#### **Input Table**

#### State\_streams Table

|  |  |
| --- | --- |
| **State** | **Total\_streams** |
| NC | 34569 |
| SC | 33999 |
| CA | 98324 |
| MA | 19345 |
| . . | . . |

#### **Output Table:**

|  |  |
| --- | --- |
| **State\_a** | **State\_b** |
| NC | SC |
| SC | NC |

### **Possible Answer** –

SELECT    a.state as state\_a,     b.state as state\_b  FROM       state\_streams a CROSS JOIN     state\_streams b WHERE     ABS(a.total\_streams – b.total\_streams) < 1000    AND     a.state <> b.state

## **OUTER JOIN**

Question – Say we have login data in the table logins. Write a query to find many users last month did not come back this month. i.e. the number of churned users.

#### **Input Table**

#### Login Table

|  |  |
| --- | --- |
| **User\_id** | **Date** |
| 1 | 2018-07-01 |
| 234 | 2018-07-02 |
| 3 | 2018-07-02 |
| 1 | 2018-07-02 |
| . . | . . |
| 234 | 2018-10-04 |

### **Possible Answer –**

SELECT    DATE\_TRUNC(‘month’, a.date) month\_timestamp,    COUNT(DISTINCT b.user\_id) churned\_users FROM     logins aFULL OUTER JOIN     logins b ON a.user\_id = b.user\_id        AND DATE\_TRUNC(‘month’, a.date) = DATE\_TRUNC(‘month’, b.date) + interval ‘1 month’WHERE     a.user\_id IS NULL GROUP BY     DATE\_TRUNC(‘month’, a.date)

# Aggregator Functions

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Practice Questions](https://dashboard.stige.in/index.php/lessons/task-7-practice-questions/) [Aggregator Functions](https://dashboard.stige.in/index.php/topic/aggregator-functions/)

In Progress

***Question : 1*** – We’re given a table of product purchases. Each row in the table represents an individual user product purchase. Write a query to get the number of customers that were upsold by purchasing additional products.

***Note*** : That if the customer purchased two things on the same day that does not count as an upsell as they were purchased within a similar timeframe.

#### **Input Table**

#### Transactions Table

|  |  |
| --- | --- |
| **Column** | **Type** |
| id | integer |
| user\_id | integer |
| created\_at | datetime |
| product\_id | integer |
| quantity | integer |

#### **Output Table**

|  |  |
| --- | --- |
| **Column** | **Type** |
| num\_of\_upsold\_customers | integer |

### **Possible Answer –**

SELECT COUNT(\*)  
FROM (  
SELECT user\_id  
FROM transactions  
GROUP BY user\_id  
HAVING COUNT(DISTINCT DATE(created\_at)) > 1  
) as t

***Question : 2*** – We have a table that represents the total number of messages sent between two users by date on messenger. Write a query to get the distribution of the number of conversations created by each user by day in the year 2020.

#### **Input Table**

#### Messages Table

|  |  |
| --- | --- |
| **Column** | **Type** |
| id | integer |
| date | date |
| user1 | integer |
| user2 | integer |
| msg\_count | integer |

#### **Output Table**

|  |  |
| --- | --- |
| **Column** | **Type** |
| num\_conversations | integer |
| frequency | integer |

### **Possible Answer –**

SELECT num\_conversations, COUNT(\*) AS frequency  
FROM (  
SELECT user1, DATE(date), COUNT(DISTINCT user2) AS num\_conversations  
FROM messages  
WHERE YEAR(date) = ‘2020’  
GROUP BY 1,2  
) AS t  
GROUP BY 1

# Windows Function

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 7: Practice Questions](https://dashboard.stige.in/index.php/lessons/task-7-practice-questions/) [Windows Function](https://dashboard.stige.in/index.php/topic/windows-function/)

In Progress

***Question : 1*** – Given the revenue transactions table above, write a query that finds the third purchase of every user.

#### **Transactions table** :

| **Column** | **Type** |
| --- | --- |
| id | integer |
| user\_id | integer |
| created\_at | datetime |
| product\_id | integer |
| quantity | integer |

#### **Output Table** :

| **Column** | **Type** |
| --- | --- |
| user\_id | integer |
| created\_at | datetime |
| product\_id | integer |
| quantity | integer |

### **Possible Answer** –

SELECT user\_id, created\_at, product\_id, quantity  
FROM (  
    SELECT  
        user\_id  
        , created\_at  
        , product\_id  
        , quantity  
        , RANK() OVER (  
            PARTITION BY user\_id  
            ORDER BY created\_at ASC  
        ) AS rank\_value  
    FROM transactions  
) AS t WHERE rank\_value = 3

***Question : 2*** – Given the employees and departments table, write a query to get the top 3 highest employee salaries by department. If the department contains less that 3 employees, the top 2 or the top 1 highest salaries should be listed (assume that each department has at least 1 employee).  The output should include the full name of the employee in one column, the department name, and the salary. The output should be sorted by department name in ascending order and salary in descending order.

#### **Input Table:**

#### Employees Table

|  |  |
| --- | --- |
| **Column** | **Type** |
| id | int |
| first\_name | string |
| last\_name | string |
| salary | int |
| department\_id | int |

#### Departments Table

|  |  |
| --- | --- |
| **Column** | **Type** |
| Id | int |
| name | string |

#### **Output Table:**

|  |  |
| --- | --- |
| **Column** | **Type** |
| employee\_name | string |
| department\_name | string |
| salary | int |

### **Possible Answer –**

WITH table1 AS(  
  SELECT  
    department\_id,  
    first\_name,  
    last\_name,  
    salary,  
    RANK() OVER (PARTITION BY department\_id ORDER BY salary) AS ranks  
  FROM employees   
)   
SELECT  
  CONCAT(t1.first\_name,’ ‘, t1.last\_name) AS employee\_name,  
  d.name AS department\_name,  
  salary  
FROM table1 t1  
LEFT JOIN departments d ON d.id=t1.department\_id  
WHERE ranks < 4  
ORDER BY department\_name ASC, salary DESC

# Task 8: SQL Mini Project

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: SQL Mini Project](https://dashboard.stige.in/index.php/lessons/task-8-sql-mini-project/)

In Progress

There are two Case studies given below , you are required to **write the SQL query and complete your assignment** at the earliest.

### **Submission Format**

**Make a pdf** of the project and submit it through the button below. **Your pdf** **should contain the following details:**

• **Project Description:** Give a brief about your project description i.e. what is this project about, how are you going to handle the things and what are the things that you are going to find out through the project.

• **Approach:** Write a short paragraph about your approach towards the project and how you have executed it.  
  
• **Tech-Stack Used:** Do mention the software and the version used while making the project (For Eg. Jupyter Notebook, etc) and mention the purpose of using it.

• **Insights:** Jot down the insights and the knowledge you gained while making the project. You need to write that what do you infer about the things. Make sure its brief and up to the point only. For Eg. If you got a graph then what do you understand by the graph, what changes can you make or what can you derive from the graph.

•**Result:**Mention what have you achieved while making the project and how do you think it has helped you.  
  
• **Drive Link:** Save your file as a “.pdf” file and upload it to your Google Drive. Mention the sharable link (**link visibility should be set to public**) in your pdf file which you will be uploading. Do not directly upload your project.

# Case Study 1

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: SQL Mini Project](https://dashboard.stige.in/index.php/lessons/task-8-sql-mini-project/) [Case Study 1](https://dashboard.stige.in/index.php/topic/question-1/)

In Progress

#### **Case Study -1: Operation Analytics**

Table Structure: job\_data

* job\_id: unique identifier of jobs
* actor\_id: unique identifier of actor
* event: decision/skip/transfer
* language: language of the content
* time\_spent: time spent to review the job in seconds
* org: organization of the actor,
* ds: date in the yyyy/mm/dd format. It is stored in the form of text and we use presto to run. no need for date function

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ds** | **job\_id** | **actor\_id** | **event** | **language** | **time\_spent** | **org** |
| 2020-11-30 | 21 | 1001 | skip | English | 15 | A |
| 2020-11-30 | 22 | 1006 | transfer | Arabic | 25 | B |
| 2020-11-29 | 23 | 1003 | decision | Persian | 20 | C |
| 2020-11-28 | 23 | 1005 | transfer | Persian | 22 | D |
| 2020-11-28 | 25 | 1002 | decision | Hindi | 11 | B |
| 2020-11-27 | 11 | 1007 | decision | French | 104 | D |
| 2020-11-26 | 23 | 1004 | skip | Persian | 56 | A |
| 2020-11-25 | 20 | 1003 | transfer | Italian | 45 | C |

Signals:

* Points to be considered :
  + What does the event mean? What to consider for reviewing?
  + Candidate should spend some time understanding the table

**QA** : **Calculate the number of jobs reviewed per hour per day for November 2020?**

**QB** : **Let’s say the above metric is called throughput. Calculate 7 day rolling average of throughput? For throughput, do you prefer daily metric or 7-day rolling and why?**

**QC** : **Calculate the percentage share of each language in the last 30 days?**

**QD**: **Let’s say you see some duplicate rows in the data. How will you display duplicates from the table?**

# Case Study 2

[Data Analytics](https://dashboard.stige.in/index.php/courses/lms-data-analytics/) [Task 8: SQL Mini Project](https://dashboard.stige.in/index.php/lessons/task-8-sql-mini-project/) [Case Study 2](https://dashboard.stige.in/index.php/topic/question-2/)

In Progress

#### **Case Study – 2: Investigating metric Spike**

You show upto work Tuesday morning, September 2, 2014. The head of the Product team walks over to your desk and asks you to investigate the dip in weekly engagement.

**Table-1: users**

 This table includes one row per user, with descriptive information about that user’s account.

|  |  |
| --- | --- |
| **user\_id** | A unique ID per user. Can be joined to user\_id in either of the other tables. |
| **created\_at** | The time the user was created (first signed up) |
| **state** | The state of the user (active or pending) |
| **activated\_at** | The time the user was activated, if they are active |
| **company\_id** | The ID of the user’s company |
| **language** | The chosen language of the user |

**Table-2: events**

This table includes one row per event, where an event is an action that a user has taken. These events include login events, messaging events, search events, events logged as users progress through a signup funnel, events around received emails.

|  |  |
| --- | --- |
| **user\_id** | The ID of the user logging the event. Can be joined to user\\_id in either of the other tables. |
| **occurred\_at** | The time the event occurred. |
| **event\_type** | The general event type. There are two values in this dataset: “signup\_flow”, which refers to anything occuring during the process of a user’s authentication, and “engagement”, which refers to general product usage after the user has signed up for the first time. |
| **event\_name** | The specific action the user took. Possible values include: create\_user: User is added to Yammer’s database during signup process enter\_email: User begins the signup process by entering her email address enter\_info: User enters her name and personal information during signup process complete\_signup: User completes the entire signup/authentication process home\_page: User loads the home page like\_message: User likes another user’s message login: User logs into Yammer search\_autocomplete: User selects a search result from the autocomplete list search\_run: User runs a search query and is taken to the search results page search\_click\_result\_X: User clicks search result X on the results page, where X is a number from 1 through 10. send\_message: User posts a message view\_inbox: User views messages in her inbox |
| **location:** | The country from which the event was logged (collected through IP address). |
| **device:** | The type of device used to log the event. |

**Table-3: email\_events**

This table contains events specific to the sending of emails. It is similar in structure to the events table above.

|  |  |
| --- | --- |
| **user\_id** | The ID of the user to whom the event relates. Can be joined to user\_id in either of the other tables. |
| **occurred\_at** | The time the event occurred. |
| **action** | The name of the event that occurred. “sent\_weekly\_digest” means that the user was delivered a digest email showing relevant conversations from the previous day. “email\_open” means that the user opened the email. “email\_clickthrough” means that the user clicked a link in the email. |

**QA : Calculate the weekly user engagement?**

**QB : Calculate the user growth for product?**

**QC : Calculate the weekly retention of users-sign up cohort?**

**QD : Calculate the weekly engagement per device?**

**QE :** **Calculate the email engagement metrics?**

Click the below button to get the dataset –

<https://drive.google.com/drive/folders/1u2oZPrCVBINZUQE4EfBprWzNfaJjHokV?usp=sharing>