# [BASIC SQL](https://community.modeanalytics.com/sql/tutorial/introduction-to-sql/)

## SQL SELECT

If you’d like your results to look a bit more presentable, you can rename columns to include spaces. For example, if you want the west column to appear as West Region in the results, you would have to type:

**SELECT** west **AS** "West Region"

**FROM** tutorial.us\_housing\_units

Note that the results will only return capital letters if you put column names in double quotes. The following query, for example, will return results with lower-case column names.

**SELECT** west **AS** West\_Region,

south **AS** South\_Region

**FROM** tutorial.us\_housing\_units

## SQL WHERE

**SELECT** \*

**FROM** tutorial.us\_housing\_units

**WHERE** **month** = 1

Note: the clauses always need to be in this order:SELECT, FROM, WHERE.

## SQL Comparison Operators

Equal to =

Not equal to <> or !=

Greater than >

Less than <

Greater than or equal to >=

Less than or equal to <=

**SELECT** \*

**FROM** tutorial.us\_housing\_units

**WHERE** west > 30

**SELECT** \*

**FROM** tutorial.us\_housing\_units

**WHERE** month\_name != 'January'

If you’re using an operator with values that are non-numeric, you need to put the value in single quotes: 'value'.

**Note:** SQL uses double quotes to reference [column names](https://community.modeanalytics.com/sql/tutorial/sql-select-statement/#column-names).

If you’re using >, <, >=, or <=, you don’t necessarily need to be too specific about how you filter. Try this:

**SELECT** \*

**FROM** tutorial.us\_housing\_units

**WHERE** month\_name > 'J'

The way SQL treats alphabetical ordering is a little bit tricky. You may have noticed in the above query that selecting month\_name > 'J' will yield only rows in which month\_name starts with “j” or later in the alphabet. “Wait a minute,” you might say. “January is included in the results—shouldn’t I have to use month\_name >= 'J to make that happen?” SQL considers ‘Ja’ to be greater than ‘J’ because it has an extra letter. It’s worth noting that most dictionaries would list ‘Ja’ after ‘J’ as well.

You can perform arithmetic in SQL using the same operators you would in Excel: +, -, \*, /.

**SELECT** **year**,

**month**,

west,

south,

west + south **AS** south\_plus\_west

**FROM** tutorial.us\_housing\_units

**SELECT** **year**,

**month**,

west,

south,

(west + south)/2 **AS** south\_west\_avg

**FROM** tutorial.us\_housing\_units

## SQL Logical Operators

* [LIKE](https://community.modeanalytics.com/sql/tutorial/sql-like/) allows you to match similar values, instead of exact values.
* [IN](https://community.modeanalytics.com/sql/tutorial/sql-in-operator/) allows you to specify a list of values you’d like to include.
* [BETWEEN](https://community.modeanalytics.com/sql/tutorial/sql-between/) allows you to select only rows within a certain range.
* [IS NULL](https://community.modeanalytics.com/sql/tutorial/sql-is-null/) allows you to select rows that contain no data in a given column.
* [AND](https://community.modeanalytics.com/sql/tutorial/sql-and-operator/) allows you to select only rows that satisfy two conditions.
* [OR](https://community.modeanalytics.com/sql/tutorial/sql-or-operator/) allows you to select rows that satisfy either of two conditions.
* [NOT](https://community.modeanalytics.com/sql/tutorial/sql-not-operator/) allows you to select rows that do not match a certain condition.

## SQL LIKE

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** "group" **LIKE** 'Snoop%'

**Note:** "group" appears in quotations above because GROUP is actually the [name of a function in SQL](https://community.modeanalytics.com/sql/tutorial/sql-aggregate-functions/). The double quotes (as opposed to single: ') are a way of indicating that you are referring to the column name "group", not the SQL function.

The % used above represents any character or set of characters. In this case, % is referred to as a “wildcard.” In the type of SQL that Mode uses, LIKE is case-sensitive, meaning that the above query will only capture matches that start with a capital “S” and lower-case “noop.” To ignore case when you’re matching values, you can use the ILIKE command:

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** "group" **ILIKE** 'snoop%'

You can also use \_ (a single underscore) to substitute for an individual character:

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** artist **ILIKE** 'dr\_ke'

## SQL IN

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** year\_rank **IN** (1, 2, 3)

You can use non-numerical values, but they need to go inside single quotes

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** artist **IN** ('Taylor Swift', 'Usher', 'Ludacris')

## SQL BETWEEN

[dataset](https://community.modeanalytics.com/sql/tutorial/sql-logical-operators/#about-this-dataset):

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** year\_rank **BETWEEN** 5 **AND** 10

Between includes the boundary points, same result as below:

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** year\_rank >= 5 **AND** year\_rank <= 10

## SQL IS NULL

You can select rows that contain no data in a given column by using IS NULL.

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** artist **IS** NULL

WHERE artist = NULL will not work—you can’t perform arithmetic on null values.

## SQL AND

the following query will return all rows for top-10 recordings in 2012.

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** **year** = 2012 **AND** year\_rank <= 10

## SQL OR

You’ll notice that each row will satisfy one of the two conditions. You can combine AND with OR using parenthesis. The following query will return rows that satisfyboth of the following conditions:

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** **year** = 2013

**AND** ("group" **ILIKE** '%macklemore%' **OR** "group" **ILIKE** '%timberlake%')

## SQL NOT

NOT is a [logical operator](https://community.modeanalytics.com/sql/tutorial/sql-logical-operators/) in SQL that you can put before any conditional statement to select rows for which that statement is false.

Using NOT with < and > usually doesn’t make sense because you can simply use the opposite comparative operator instead. For example, this query will return an error:

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** **year** = 2013

**AND** year\_rank **NOT** > 3

Instead, you would just write that as:

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** **year** = 2013

**AND** year\_rank <= 3

NOT is commonly used with LIKE. Run this query and check out how Macklemore magically disappears!

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** **year** = 2013

**AND** "group" **NOT** **ILIKE** '%macklemore%'

## SQL ORDER BY

The ORDER BY clause allows you to reorder your results based on the data in one or more columns.

You’ll notice that the results are now ordered alphabetically from a to z based on the content in the artist column. This is referred to as ascending order, and it’s SQL’s default. If you order a numerical column in ascending order, it will start with smaller (or most negative) numbers

## Ordering data by multiple columns

You can also order by mutiple columns. This example query makes the most recent years come first but orders top-ranks songs before lower-ranked songs:

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** year\_rank <= 3

**ORDER** **BY** **year** **DESC**, year\_rank

First, columns in the ORDER BY clause must be separated by commas. Second, the DESC operator is only applied to the column that precedes it.

you can make your life a little easier by substituting numbers for column names in the ORDER BY clause. The numbers will correspond to the order in which you list columns in the SELECT clause.

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** year\_rank <= 3

**ORDER** **BY** 2, 1 **DESC**

When using ORDER BY with a row limit (either through the check box on the query editor or by typing in LIMIT), the ordering clause is executed first. This means that the results are ordered before limiting to only a few rows.

## Using comments

You can use-- (two dashes) to comment out everything to the right of them on a given line:

**SELECT** \* --This comment won't affect the way the code runs

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** **year** = 2013

You can also leave comments across multiple lines using /\*to begin the comment and \*/ to close it:

/\* Here's a comment so long and descriptive that

it could only fit on multiple lines. Fortunately,

it, too, will not affect how this code runs. \*/

**SELECT** \*

**FROM** tutorial.billboard\_top\_100\_year\_end

**WHERE** **year** = 2013

# [INTERMEDIATE SQL](https://community.modeanalytics.com/sql/tutorial/sql-aggregate-functions/)

## SQL Aggregate Functions

## SQL COUNT

COUNT is a [SQL aggregate function](https://community.modeanalytics.com/sql/tutorial/sql-aggregate-functions/) for counting the number of rows in a particular column.

**SELECT** **COUNT**(\*)

**FROM** tutorial.aapl\_historical\_stock\_price

*Note: Typing* COUNT(1) *has the same effect as* COUNT(\*)*. Which one you use is a matter of personal preference.*

Things start to get a little bit tricky when you want to count individual columns. The following code will provide a count of all of rows in which the high column is not null.

**SELECT** **COUNT**(**high**)

**FROM** tutorial.aapl\_historical\_stock\_price

You can add column names (also called *aliases*) using AS:

**SELECT** **COUNT**(date) **AS** count\_of\_date

**FROM** tutorial.aapl\_historical\_stock\_price

## SQL SUM

SUM is a [SQL aggregate function](https://community.modeanalytics.com/sql/tutorial/sql-aggregate-functions/) that totals the values in a given column. Unlike [COUNT](https://community.modeanalytics.com/sql/tutorial/sql-count/), you can only use SUM on columns containing numerical values.

The query below selects the sum of the volume column from the [Apple stock prices dataset](https://community.modeanalytics.com/sql/tutorial/sql-aggregate-functions/#the-apple-stock-prices-dataset):

**SELECT** **SUM**(volume)

**FROM** tutorial.aapl\_historical\_stock\_price

An important thing to remember: **aggregators only aggregate vertically**. If you want to perform a calculation across rows, you would do this with [simple arithmetic](https://community.modeanalytics.com/sql/tutorial/sql-operators/#arithmetic-in-sql).

SELECT SUM(open)/COUNT(open) AS avg\_open\_price

FROM tutorial.aapl\_historical\_stock\_price

## SQL MIN/MAX

MIN and MAX are [SQL aggregation functions](https://community.modeanalytics.com/sql/tutorial/sql-aggregate-functions/) that return the lowest and highest values in a particular column.

They’re similar to [COUNT](https://community.modeanalytics.com/sql/tutorial/sql-count/) in that they can be used on non-numerical columns. Depending on the column type, MIN will return the lowest number, earliest date, or non-numerical value as close alphabetically to “A” as possible. As you might suspect, MAX does the opposite—it returns the highest number, the latest date, or the non-numerical value closest alphabetically to “Z.”

.

**SELECT** **MIN**(volume) **AS** min\_volume,

**MAX**(volume) **AS** max\_volume

**FROM** tutorial.aapl\_historical\_stock\_price

## SQL AVG

AVG is a [SQL aggregate function](https://community.modeanalytics.com/sql/tutorial/sql-aggregate-functions/) that calculates the average of a selected group of values. It’s very useful, but has some limitations. First, it can only be used on numerical columns. Second, it ignores nulls completely.

**SELECT** **AVG**(**high**)

**FROM** tutorial.aapl\_historical\_stock\_price

**WHERE** **high** **IS** **NOT** NULL

The above query produces the same result as the following query:

**SELECT** **AVG**(**high**)

**FROM** tutorial.aapl\_historical\_stock\_price

## SQL GROUP BY

[SQL aggregate functions](https://community.modeanalytics.com/sql/tutorial/sql-aggregate-functions/) like COUNT, AVG, and SUM have something in common: they all aggregate across the entire table.