**SQL**

**1**

A relational database is a database that organizes information into one or more tables.

A table is a collection of data organized into rows and columns. Tables are sometimes referred to as relations.

A column is a set of data values of a particular type. Here, id, name, and age are the columns.

|  |  |  |
| --- | --- | --- |
| ID | NAME | AGE |
| 1 | Juan | 30 |
| 2 | Noe | 29 |

A row is a single record in a table. The first row in the celebs table has:

* An id of 1
* A name of Juan
* An age of 30

All data stored in a relational database is of a certain data type. Some of the most common data types are:

* INTEGER, a positive or negative whole number (-1 or 1, etc)
* TEXT, a text string (“Juan”)
* DATE, the date formatted as YYYY-MM-DD (2022-01-20)
* REAL, a decimal value (1.00)

**Statements**

The code below is a SQL statement. A *statement* is text that the database recognizes as a valid command. Statements always end in a semicolon ;.

Texto

Descripción generada automáticamente

Componentes de un statement:

* 1. Clause: ***(commands)*** Clauses perform specific tasks in SQL. By convention, clauses are written in capital letters. Clauses can also be referred to as commands. CREATE TABLE
  2. Table name: refers to the name of the table that the command is applied to. Table\_name
  3. Parameters: A parameter is a list of columns, data types, or values that are passed to a clause as an argument. Here, the parameter is a list of column names and the associated data type. (column\_1 data\_type, column\_2 data\_type, column\_3 data\_type)

**Create**

CREATE statements allow us to create a new table in the database. You can use the CREATE statement anytime you want to create a new table from scratch. The statement below creates a new table named celebs.

Un conjunto de letras blancas en un fondo blanco

Descripción generada automáticamente con confianza media

1. CREATE TABLE is a clause that tells SQL you want to create a new table.  
2. celebs is the name of the table.  
3. (id INTEGER, name TEXT, age INTEGER) is a list of parameters defining each column, or attribute in the table and its data type:

* id is the first column in the table. It stores values of data type INTEGER
* name is the second column in the table. It stores values of data type TEXT
* age is the third column in the table. It stores values of data type INTEGER

**Insert**

The INSERT statement inserts a new row into a table.

We can use the INSERT statement when you want to add new records. The statement below enters a record for Justin Bieber into the celebs table.

Texto

Descripción generada automáticamente

* INSERT INTO is a clause that adds the specified row or rows.
* celebs is the table the row is added to.
* (id, name, age) is a parameter identifying the columns that data will be inserted into.
* VALUES is a clause that indicat es the data being inserted.
* (1, 'Justin Bieber', 22) is a parameter identifying the values being inserted.
  + 1: an integer that will be added to id column
  + 'Justin Bieber': text that will be added to name column
  + 22: an integer that will be added to age column

**Select**

SELECT statements are used to fetch data from a database. In the statement below, SELECT returns all data in the name column of the celebs table.

Imagen que contiene Texto

Descripción generada automáticamente

1. SELECT is a clause that indicates that the statement is a query. You will use SELECT every time you query data from a database.

2. name specifies the column to query data from.

3. FROM celebs specifies the name of the table to query data from. In this statement, data is queried from the celebs table.

You can also query data from all columns in a table with SELECT.

Forma, Rectángulo

Descripción generada automáticamente

\* is a special wildcard character that we have been using. It allows you to select every column in a table without having to name each one individually. Here, the result set contains every column in the celebs table.

SELECT statements always return a new table called the *result set*.

**Alter**

The ALTER TABLE statement adds a new column to a table. You can use this command when you want to add columns to a table. The statement below adds a new column twitter\_handle to the celebs table.

Texto

Descripción generada automáticamente

1. ALTER TABLE is a clause that lets you make the specified changes.  
2. celebs is the name of the table that is being changed.  
3. ADD COLUMN is a clause that lets you add a new column to a table:

* twitter\_handle is the name of the new column being added
* TEXT is the data type for the new column

4. NULL is a special value in SQL that represents missing or unknown data. Here, the rows that existed before the column was added have NULL (∅) values for twitter\_handle.

Tabla

Descripción generada automáticamente

**Update**

The UPDATE statement edits a row in a table. You can use the UPDATE statement when you want to change existing records. The statement below updates the record with an id value of 4 to have the twitter\_handle @taylorswift13.

Texto

Descripción generada automáticamente

1. UPDATE is a clause that edits a row in the table.  
2. celebs is the name of the table.  
3. SET is a clause that indicates the column to edit.

* twitter\_handle is the name of the column that is going to be updated
* @taylorswift13 is the new value that is going to be inserted into the twitter\_handle column.

4. WHERE is a clause that indicates which row(s) to update with the new column value. Here the row with a 4 in the id column is the row that will have the twitter\_handle updated to @taylorswift13.

Tabla

Descripción generada automáticamente

**Delete**

The DELETE FROM statement deletes one or more rows from a table. You can use the statement when you want to delete existing records. The statement below deletes all records in the celeb table with no twitter\_handle:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. DELETE FROM is a clause that lets you delete rows from a table.
2. celebs is the name of the table we want to delete rows from.
3. WHERE is a clause that lets you select which rows you want to delete. Here we want to delete all of the rows where the twitter\_handle column IS NULL.
4. IS NULL is a condition in SQL that returns true when the value is NULL and false otherwise.

Calendario

Descripción generada automáticamente

**Constraints**

*Constraints* that add information about how a column can be used are invoked after specifying the data type for a column. They can be used to tell the database to reject inserted data that does not adhere to a certain restriction. The statement below sets *constraints* on the celebs table.

Texto

Descripción generada automáticamente

1. PRIMARY KEY columns can be used to uniquely identify the row. Attempts to insert a row with an identical value to a row already in the table will result in a *constraint violation* which will not allow you to insert the new row.

2. UNIQUE columns have a different value for every row. This is similar to PRIMARY KEY except a table can have many different UNIQUE columns.

3. NOT NULL columns must have a value. Attempts to insert a row without a value for a NOT NULL column will result in a constraint violation and the new row will not be inserted.

4. DEFAULT columns take an additional argument that will be the assumed value for an inserted row if the new row does not specify a value for that column.

**QUERIES**

**Introduction**

In this lesson, we will be learning different SQL commands to **query** a single table in a database.

One of the core purposes of the SQL language is to retrieve information stored in a database. This is commonly referred to as querying. Queries allow us to communicate with the database by asking questions and returning a result set with data relevant to the question.

We will be querying a database with one table named movies.

Let’s get started!

**Select**

Previously, we learned that SELECT is used every time you want to query data from a database and \* means *all* columns.

Suppose we are only interested in two of the columns. We can select individual columns by their names (separated by a comma):

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

To make it easier to read, we moved FROM to another line.

Line breaks don’t mean anything specific in SQL. We could write this entire query in one line, and it would run just fine.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

**As**

Knowing how SELECT works, suppose we have the code below:

Imagen que contiene Texto

Descripción generada automáticamente

Can you guess what AS does?

AS is a keyword in SQL that allows you to rename a column or table using an alias. The new name can be anything you want as long as you put it inside of single quotes. Here we renamed the name column as Titles.

Some important things to note:

* Although it’s not always necessary, it’s best practice to surround your aliases with single quotes.
* When using AS, the columns are not being renamed in the table. The aliases only appear in the result.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

**Distinct**

When we are examining data in a table, it can be helpful to know what *distinct* values exist in a particular column.

DISTINCT is used to return unique values in the output. It filters out all duplicate values in the specified column(s).

For instance,

Forma, Rectángulo

Descripción generada automáticamente

Might Produce

Patrón de fondo

Descripción generada automáticamente

By adding DISTINCT before the column name,

Texto

Descripción generada automáticamente con confianza media

the result would now be:

Tabla

Descripción generada automáticamente con confianza media

Filtering the results of a query is an important skill in SQL. It is easier to see the different possible genres in the movie table after the data has been filtered than to scan every row in the table.

**Where**

We can restrict our query results using the WHERE clause in order to obtain only the information we want.

Following this format, the statement below filters the result set to only include top rated movies (IMDb ratings greater than 8):

Imagen que contiene Rectángulo

Descripción generada automáticamente

How does it work?

1. WHERE clause filters the result set to only include rows where the following condition is true.
2. imdb\_rating > 8 is the condition. Here, only rows with a value greater than 8 in the imdb\_rating column will be returned.

The > is an operator. Operators create a condition that can be evaluated as either true or false.

Comparison operators used with the WHERE clause are:

* = equal to
* != not equal to
* > greater than
* < less than
* >= greater than or equal to
* <= less than or equal to

**Like I**

LIKE can be a useful operator when you want to compare similar values.

The movies table contains two films with similar titles, ‘Se7en’ and ‘Seven’.

How could we select all movies that start with ‘Se’ and end with ‘en’ and have exactly one character in the middle?

Texto

Descripción generada automáticamente con confianza media

* LIKE is a special operator used with the WHERE clause to search for a specific pattern in a column.
* name LIKE 'Se\_en' is a condition evaluating the name column for a specific pattern.
* Se\_en represents a pattern with a wildcard character.

The \_ means you can substitute any individual character here without breaking the pattern. The names Seven and Se7en both match this pattern.

The percentage sign % is another wildcard character that can be used with LIKE.

This statement below filters the result set to only include movies with names that begin with the letter ‘A’:

Imagen que contiene Rectángulo

Descripción generada automáticamente

% is a wildcard character that matches zero or more missing letters in the pattern. For example:

* A% matches all movies with names that begin with letter ‘A’
* %a matches all movies that end with ‘a’

We can also use % both before and after a pattern:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Here, any movie that *contains* the word ‘man’ in its name will be returned in the result.

LIKE is not case sensitive. ‘Batman’ and ‘Man of Steel’ will both appear in the result of the query above.

**Is Null**

By this point of the lesson, you might have noticed that there are a few missing values in the movies table. More often than not, the data you encounter will have missing values.

Unknown values are indicated by NULL.

It is not possible to test for NULL values with comparison operators, such as = and !=.

Instead, we will have to use these operators:

* IS NULL
* IS NOT NULL

To filter for all movies *with* an IMDb rating:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

**Between**

The BETWEEN operator is used in a WHERE clause to filter the result set within a certain *range*. It accepts two values that are either numbers, text or dates.

For example, this statement filters the result set to only include movies with years from 1990 up to, *and including* 1999.

Diagrama

Descripción generada automáticamente con confianza media

When the values are text, BETWEEN filters the result set for within the alphabetical range.

In this statement, BETWEEN filters the result set to only include movies with names that begin with the letter ‘A’ up to, *but not including* ones that begin with ‘J’.

Texto

Descripción generada automáticamente con confianza media

However, if a movie has a name of simply ‘J’, it would actually match. This is because BETWEEN goes up to the second value — up to ‘J’. So the movie named ‘J’ would be included in the result set but not ‘Jaws’.

**And**

Sometimes we want to *combine multiple conditions* in a WHERE clause to make the result set more specific and useful.

One way of doing this is to use the AND operator. Here, we use the AND operator to only return 90’s romance movies.

Texto

Descripción generada automáticamente

* year BETWEEN 1990 AND 1999 is the 1st condition.
* genre = 'romance' is the 2nd condition.
* AND combines the two conditions.

Diagrama, Diagrama de Venn

Descripción generada automáticamente

With AND, both conditions must be true for the row to be included in the result.

**Or**

Similar to AND, the OR operator can also be used to combine multiple conditions in WHERE, but there is a fundamental difference:

* AND operator displays a row if *all* the conditions are true.
* OR operator displays a row if *any* condition is true.

Suppose we want to check out a new movie or something action-packed:

Texto

Descripción generada automáticamente con confianza media

* year > 2014 is the 1st condition.
* genre = 'action' is the 2nd condition.
* OR combines the two conditions.

Diagrama, Diagrama de Venn

Descripción generada automáticamente

With OR, if any of the conditions are true, then the row is added to the result.

**Order By**

That’s it with WHERE and its operators. Moving on!

It is often useful to list the data in our result set in a particular order.

We can *sort* the results using ORDER BY, either alphabetically or numerically. Sorting the results often makes the data more useful and easier to analyze.

For example, if we want to sort everything by the movie’s title from A through Z:

Forma, Rectángulo

Descripción generada automáticamente

* ORDER BY is a clause that indicates you want to sort the result set by a particular column.
* name is the specified column.

Sometimes we want to sort things in a decreasing order. For example, if we want to select all of the well-received movies, sorted from highest to lowest by their year:

Texto

Descripción generada automáticamente

* DESC is a keyword used in ORDER BY to sort the results in descending order (high to low or Z-A).
* ASC is a keyword used in ORDER BY to sort the results in ascending order (low to high or A-Z).

The column that we ORDER BY doesn’t even have to be one of the columns that we’re displaying.

Note: ORDER BY always goes after WHERE (if WHERE is present).

**Limit**

We’ve been working with a fairly small table (fewer than 250 rows), but most SQL tables contain hundreds of thousands of records. In those situations, it becomes important to cap the number of rows in the result.

For instance, imagine that we just want to see a few examples of records.

Forma, Rectángulo

Descripción generada automáticamente

LIMIT is a clause that lets you specify the maximum number of rows the result set will have. This saves space on our screen and makes our queries run faster.

Here, we specify that the result set can’t have more than 10 rows.

LIMIT always goes at the very end of the query. Also, it is not supported in all SQL databases.

**Case**

A CASE statement allows us to create different outputs (usually in the SELECT statement). It is SQL’s way of handling [if-then](https://en.wikipedia.org/wiki/Conditional_(computer_programming)" \t "_blank) logic.

Suppose we want to condense the ratings in movies to three levels:

* *If the rating is above 8, then it is Fantastic.*
* *If the rating is above 6, then it is Poorly Received.*
* *Else, Avoid at All Costs.*

Texto

Descripción generada automáticamente

* Each WHEN tests a condition and the following THEN gives us the string if the condition is true.
* The ELSE gives us the string if all the above conditions are false.
* The CASE statement must end with END.

In the result, you have to scroll right because the column name is very long. To shorten it, we can rename the column to ‘Review’ using AS:

Texto

Descripción generada automáticamente

**Aggregate**

Calculations performed on multiple rows of a table are called **aggregates**.

In this lesson, we have given you a table named fake\_apps which is made up of fake mobile applications data.

Here is a quick preview of some important aggregates that we will cover in the next five exercises:

* COUNT(): count the number of rows
* SUM(): the sum of the values in a column
* MAX()/MIN(): the largest/smallest value
* AVG(): the average of the values in a column
* ROUND(): round the values in the column

Let’s get started!

**Count**

The fastest way to calculate how many rows are in a table is to use the COUNT() function.

COUNT() is a function that takes the name of a column as an argument and counts the number of non-empty values in that column.

Forma, Rectángulo

Descripción generada automáticamente

Here, we want to count every row, so we pass \* as an argument inside the parenthesis.

**Sum**

SQL makes it easy to add all values in a particular column using SUM().

SUM() is a function that takes the name of a column as an argument and returns the sum of all the values in that column.

What is the total number of downloads for all of the apps combined?

Forma, Rectángulo

Descripción generada automáticamente

This adds all values in the downloads column.

**Max / Min**

The MAX() and MIN() functions return the highest and lowest values in a column, respectively.

How many downloads does the most popular app have?

Forma, Rectángulo

Descripción generada automáticamente

The most popular app has 31,090 downloads!

MAX() takes the name of a column as an argument and returns the largest value in that column. Here, we returned the largest value in the downloads column.

MIN() works the same way but it does the exact opposite; it returns the smallest value.

**Average**

SQL uses the AVG() function to quickly calculate the average value of a particular column.

The statement below returns the average number of downloads for an app in our database:

Forma, Rectángulo

Descripción generada automáticamente

The AVG() function works by taking a column name as an argument and returns the average value for that column.

**Round**

By default, SQL tries to be as precise as possible without rounding. We can make the result table easier to read using the ROUND() function.

ROUND() function takes two arguments inside the parenthesis:

1. a column name
2. an integer

It rounds the values in the column to the number of decimal places specified by the integer.

Forma, Rectángulo

Descripción generada automáticamente

Here, we pass the column price and integer 0 as arguments. SQL rounds the values in the column to 0 decimal places in the output.

**Group By I**

Oftentimes, we will want to calculate an aggregate for data with certain characteristics.

For instance, we might want to know the mean IMDb ratings for all movies each year. We could calculate each number by a series of queries with different WHERE statements, like so:

Pantalla negra con letras blancas

Descripción generada automáticamente

and so on.

Luckily, there’s a better way!

We can use GROUP BY to do this in a single step:

Forma, Rectángulo

Descripción generada automáticamente

GROUP BY is a clause in SQL that is used with aggregate functions. It is used in collaboration with the SELECT statement to arrange identical data into *groups*.

The GROUP BY statement comes after any WHERE statements, but before ORDER BY or LIMIT.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

**Group By II**

Sometimes, we want to GROUP BY a calculation done on a column.

For instance, we might want to know how many movies have IMDb ratings that round to 1, 2, 3, 4, 5. We could do this using the following syntax:

Texto

Descripción generada automáticamente con confianza media

However, this query may be time-consuming to write and more prone to error.

SQL lets us use column reference(s) in our GROUP BY that will make our lives easier.

* 1 is the first column selected
* 2 is the second column selected
* 3 is the third column selected

and so on.

The following query is equivalent to the one above:

Forma

Descripción generada automáticamente

Here, the 1 refers to the first column in our SELECT statement, ROUND(imdb\_rating).

**Having**

In addition to being able to group data using GROUP BY, SQL also allows you to filter which groups to include and which to exclude.

For instance, imagine that we want to see how many movies of different genres were produced each year, but we only care about years and genres with at least 10 movies.

We can’t use WHERE here because we don’t want to filter the rows; we want to *filter groups*.

This is where HAVING comes in.

HAVING is very similar to WHERE. In fact, all types of WHERE clauses you learned about thus far can be used with HAVING.

We can use the following for the problem:

Forma

Descripción generada automáticamente con confianza media

* When we want to limit the results of a query based on values of the individual rows, use WHERE.
* When we want to limit the results of a query based on an aggregate property, use HAVING.

HAVING statement always comes after GROUP BY, but before ORDER BY and LIMIT.

Texto

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente con confianza media