SQL Command

Data Types
CREATE TABLE

INSERT INTO
SELECT
* (ALL)

UPDATE
ALTER TABLE
DELETE FROM
IS NULL
SELECT DISTINCT
WHERE

LIKE

BETWEEN

OR

ORDER BY

LIMIT
Working with Aggregate Functions (SQL):

COUNT ()

GROUP BY

SUM()

MAX()

MIN()

AVG()

ROUND()

multi-table SQL

PRIMARY KEY

foreign key (not a command)

cross join (not a command)

inner join (not a command)

outer join (not a command)

AS

Function

All data stored in a relational database is of a certain data type
creates a table with a given name and columns

adds the specified row(s) to the selected table
used to fetch data from the database
allows us to select all columns from a given table

edits a row in the table
allows us to make specified changes to the selected table
delets one or more rows from the specified table
a conditional in SQL that returns a boolean
used to return unique values
can be used to filter results

a special operator that can be used with where to find a specific pattern in a column

allows us to filter our results to a specified range using dates, letters, integers, etc

can be used with WHERE to combine more than one condition
allows us to sort the results of our query by DESC (Z-A or high to low) or by ASC (A-Z or low to high)

limits the number of results returned to a specified number

COUNT() is a function that takes a column name and counts the number of rows where the column is not NULL

an operator that is only used with aggregate functions with SELECT to arrange identical data into groups it is common to GROUP BY the same column you SELECT

SUM() is a function that takes a column name and allows us to add values from that specific column

MAX() is a function that finds the largest number in a specified column

MIN() is a function that finds the smallest number in a specified column

AVG() is a function that takes a column name and finds the average of the numbers within that column

ROUND() takes a column name and an integer as an argument and rounds the values in the specified column to the specified number of digits after the decimal

serves as a unique identifier for each row or record in a given table

A *foreign key* is a column that contains the primary key of another table in the database. We use foreign keys and primary keys to connect rows in two different tables. One table's foreign key holds the value of another table's primary key. Unlike primary keys, foreign keys do not need to be unique and can be NULL

One way to query multiple tables is to write aSELECT statement with multiple table names separated by a comma. This is also known as a *cross join*. Here, albums and artists are the different tables we are querying.

In SQL, joins are used to combine rows from two or more tables.
The most common type of join in SQL is an *inner join*.
An inner join will combine rows from different tables if the join condition is true. Let's look at the syntax to see how it works.

Outer joins also combine rows from two or more tables, but unlike inner joins, they do not require the join condition to be met. Instead, every row in the *left* table is returned in the result set, and if the join condition is not met, then NULL values are used to fill in the columns from the *right* table.

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

Generic

```
CREATE TABLE table_name(
   column_1 data_type,
   column_2 data_type,
);

INSERT INTO table_name (column_1, column_2, column_3) VALUES (data_1, data_2, data_3);
SELECT column FROM table_name
SELECT * FROM table_name
UPDATE table_name
SET column = new_data
WHERE column = static_data;
ALTER TABLE table_name ADD COLUMN column_name data_type;
DELETE FROM table_name WHERE column_name IS NULL;

SELECT DISTINCT column_name FROM table_name;
SELECT * FROM table_name WHERE column_name condition
```

```
SELECT * FROM table_name
WHERE column_name LIKE "pattern"
```

```
SELECT * FROM table_name
WHERE column_name BETWEEN range_1 AND range_2;
SELECT * FROM table_name
WHERE condition_1
OR condition_2;
SELECT * FROM table_name
ORDER BY column_name order
SELECT * FROM table_name
ORDER BY column_name order
LIMIT quantity;
```

```
SELCET COUNT(column_name) FROM table_name;
```

```
   SELECT column_name COUNT(*) FROM table_name
   GROUP BY column_name;
```

```
SELECT SUM(column_name) FROM table_name;
```

```
SELECT MAX(column_name) FROM table_name;
```

```
SELECT MIN(column_name) FROM table_name;
```

```
SELECT AVG(column_name) FROM table_name;
```

```
SELECT column_name1, ROUND(AVG(column_name2)) FROM table_name
GROUP BY column_name;
```

```
CREATE TABLE(column_1 data_type PRIMARY KEY, column_2 data_type);
```

Here, artist_id is a foreign key in the albums table. We can see that Michael Jackson has an id of 3 in theartists table. All of the albums by Michael Jackson also have a 3 in the artist_id column in the albums table.

When querying more than one table, column names need to be specified by table_name.column_name. Here, the result set includes the name and yearcolumns from the albums table and the namecolumn from the artists table.

1. SELECT * specifies the columns our result set will have. Here, we want to include every column in both tables.
2. FROM albums specifies the first table we are querying.
3. JOIN artists ON specifies the type of join we are going to use as well as the name of the second table. Here, we want to do an inner join and the second table we want to query is artists.
4. albums.artist_id = artists.id is the join condition that describes how the two tables are related to each other. Here, SQL uses the foreign key column artist_id in the albums table to match it with exactly one row in the artists table with the same value in the id column. We know it will only match one row in the artists table because id is the PRIMARY KEY of artists.

The left table is simply the first table that appears in the statement. Here, the left table is albums. Likewise, the right table is the second table that appears. Here,artists is the right table.

It is important to note that the columns have not been renamed in either table. The aliases only appear in the result set.

eg.

TEXT, INTEGER, DATE, REAL

```
CREATE TABLE celebs(
  id INTEGER,
  name TEXT,
  age INTEGER);

INSERT INTO celebs (id, name, age) VALUES (1, 'Justin Bieber', 21);
SELECT name FROM celebs
SELECT * FROM celebs
UPDATE celebs
SET age = 22
WHERE id = 1;
ALTER TABLE celebs ADD COLUMN twitter_handle TEXT;
DELETE FROM celebs WHERE twitter_handle IS NULL

SELECT DISTINCT genre FROM movies;
SELECT * FROM movies WHERE imdb_rating > 8;
```

```
SELECT * FROM movies
WHERE name LIKE "Se_en";
```

```
SELECT * FROM movies
WHERE year BETWEEN 1999 AND 2011;
SELECT * FROM movies
WHERE genre = "comedy"
OR year < 1980;
SELECT * FROM movies
ORDER BY imdb_rating DESC
SELECT * FROM movies
ORDER BY imdb_rating DESC
LIMIT 3;
```

```
SELECT COUNT(*) FROM fake_apps
   SELECT price COUNT(*) FROM fake_apps
   WHERE downloads < 20000
   GROUP BY price;
```

```
SELECT SUM(downloads) FROM fake_apps;
```

```
SELECT MAX(downloads) FROM fake_apps;
```

```
SELECT MIN(downloads) FROM fake_apps;
```

```
SELECT AVG(downloads) FROM fake_apps;
```

```
SELECT price, ROUND(AVG(downloads)) FROM fake_apps
GROUP BY price;
```

```
CREATE TABLE(id INTEGER PRIMARY KEY, name TEXT);
```

This is how SQL is linking data between the two tables.
The *relationship* between the artists table and thealbums table is
the id value of the artists.
Unfortunately the result of this cross join is not very useful. It combines every row of the artists table with every row of the albums table. It would be more useful to only combine the rows where the album was created by the artist.

extras

?-- the "_" is a wildcard operator that allows any character to fit in this space, so:

? --> "Se_en" returns both "Seven" and "Se7en"

?-- the "%" is a wildcard operator that matches zero or more letters in the pattern so:

?-- "a%" would return all names beginning with a

?-- "%a" would return all names ending with a

?-- "%man%" would return all names containing the string "man" anywhere in the name

can also be used with AND to further filter

```
SELECT * FROM movies
WHERE year BETWEEN 1999 AND 2011
AND genre = 'comedy';
```

can be paired with other sorting operations, so:

?-- the code in the example block just returns a total count of all rows in this table

```
   SELECT price COUNT(*) FROM fake_apps
   GROUP BY price;
```

?-- the above code would return a list of the prices and how many fall into each price catagory

can also be used with GROUP BY:

```
SELECT category, SUM(downloads) FROM fake_apps
GROUP BY category;
```

?-- the above code would return the total number of downloads sorted by category

To return the names of the most downloaded apps in each category we could query:

```
SELECT name, category, MAX(downloads) FROM fake_apps
GROUP BY category;
```

Can be used with GROUP BY as well and the values can be rounded, the following will round to the second decimal place:

```
SELECT price, ROUND(AVG(downloads), 2) FROM fake_apps
GROUP BY price;
```

Using round without providing an integer will result in the value being rounded to the nearest whole number

Specifiying a column as the PRIMARY KEY means:

?--none of the values in this column are NULL

?--each value in the column is unique

To select data from both tables we would query:

```
SELECT albums.name, albums.year, artists.name FROM albums, artists;
```

```
SELECT albums.name, albums.year, artists.name FROM albums, artists
```

```
SELECT
    *
FROM
    albums
JOIN artists ON
    albums.artist_id = artists.id
SELECT
    *
FROM
    albums
LEFT JOIN artists ON
    albums.artist_id = artists.id
SELECT
  albums.name AS 'Album',
  albums.year,
  artists.name AS 'Artist'
FROM
  albums
JOIN artists ON
  albums.artist_id = artists.id
WHERE
  albums.year > 1980;
```