## List of SQL Commands

### SELECT

SELECT is probably the most commonly-used SQL statement. You'll use it pretty much every time you query data with SQL. It allows you to define what data you want your query to return.

For example, in the code below, we’re selecting a column called name from a table called customers.

SELECT nameFROM customers;

### SELECT \*

SELECT used with an asterisk (\*) will return all of the columns in the table we're querying.

SELECT \* FROM customers;

### SELECT DISTINCT

SELECT DISTINCT only returns data that is distinct — in other words, if there are duplicate records, it will return only one copy of each.

The code below would return only rows with a unique name from the customers table.

SELECT DISTINCT nameFROM customers;

### SELECT TOP

SELECT TOP only returns the top x number or percent from a table.

The code below would return the top 50 results from the customers table:

SELECT TOP 50 \* FROM customers;

The code below would return the top 50 percent of the customers table:

SELECT TOP 50 PERCENT \* FROM customers;

### AS

AS renames a column or table with an alias that we can choose. For example, in the code below, we’re renaming the name column as first\_name:

SELECT name AS first\_nameFROM customers;

### FROM

FROM specifies the table we're pulling our data from:

SELECT nameFROM customers;

### WHERE

WHERE filters your query to only return results that match a set condition. We can use this together with conditional operators like =, >, <, >=, <=, etc.

SELECT nameFROM customersWHERE name = ‘Bob’;

### AND

AND combines two or more conditions in a single query. All of the conditions must be met for the result to be returned.

SELECT nameFROM customersWHERE name = ‘Bob’ AND age = 55;

### OR

OR combines two or more conditions in a single query. Only one of the conditions must be met for a result to be returned.

SELECT nameFROM customersWHERE name = ‘Bob’ OR age = 55;

### BETWEEN

BETWEEN filters your query to return only results that fit a specified range.

SELECT nameFROM customersWHERE age BETWEEN 45 AND 55;

### LIKE

LIKE searches for a specified pattern in a column. In the example code below, any row with a name that included the characters Bob would be returned.

SELECT nameFROM customersWHERE name LIKE ‘%Bob%’;

Other operators for LIKE:

* %x — will select all values that begin with x
* %x% — will select all values that include x
* x% — will select all values that end with x
* x%y — will select all values that begin with x and end with y
* \_x% — will select all values have x as the second character
* x\_% — will select all values that begin with x and are at least two characters long. You can add additional \_ characters to extend the length requirement, i.e. x\_\_\_%

### IN

IN allows us to specify multiple values we want to select for when using the WHERE command.

SELECT nameFROM customersWHERE name IN (‘Bob’, ‘Fred’, ‘Harry’);

### IS NULL

IS NULL will return only rows with a NULL value.

SELECT nameFROM customersWHERE name IS NULL;

### IS NOT NULL

IS NOT NULL does the opposite — it will return only rows without a NULL value.

SELECT nameFROM customersWHERE name IS NOT NULL;

### CREATE

CREATE can be used to set up a database, table, index or view.

### CREATE DATABASE

CREATE DATABASE creates a new database, assuming the user running the command has the correct admin rights.

CREATE DATABASE dataquestDB;

### CREATE TABLE

CREATE TABLE creates a new table inside a database. The terms **int** and **varchar(255)** in this example specify the datatypes of the columns we're creating.

CREATE TABLE customers (

customer\_id int,

name varchar(255),

age int);

### DROP

DROP statements can be used to delete entire databases, tables or indexes.

It goes without saying that the DROP command should only be used where absolutely necessary.

### DROP DATABASE

DROP DATABASE deletes the entire database including all of its tables, indexes etc as well as all the data within it.

Again, this is a command we want to be very, very careful about using!

DROP DATABASE dataquestDB;

### DROP TABLE

DROP TABLE deletes a table as well as the data within it.

DROP TABLE customers;

### DROP INDEX

DROP INDEX deletes an index within a database.

DROP INDEX idx\_name;

### UPDATE

The UPDATE statement is used to update data in a table. For example, the code below would update the age of any customer named Bob in the customers table to 56.

UPDATE customersSET age = 56WHERE name = ‘Bob’;

### DELETE

DELETE can remove all rows from a table (using ), or can be used as part of a WHERE clause to delete rows that meet a specific condition.

DELETE FROM customersWHERE name = ‘Bob’;

### ALTER TABLE

ALTER TABLE allows you to add or remove columns from a table. In the code snippets below, we’ll add and then remove a column for surname. The text varchar(255) specifies the datatype of the column.

ALTER TABLE customersADD surname varchar(255);

ALTER TABLE customersDROP COLUMN surname;

### AGGREGATE FUNCTIONS (COUNT/SUM/AVG/MIN/MAX)

An aggregate function performs a calculation on a set of values and returns a single result.

### COUNT

COUNT returns the number of rows that match the specified criteria. In the code below, we’re using \*, so the total row count for customers would be returned.

SELECT COUNT(\*)FROM customers;

### SUM

SUM returns the total sum of a numeric column.

SELECT SUM(age)FROM customers;

### AVG

AVG returns the average value of a numeric column.

SELECT AVG(age)FROM customers;

### MIN

MIN returns the minimum value of a numeric column.

SELECT MIN(age)FROM customers;

### MAX

MAX returns the maximum value of a numeric column.

SELECT MAX(age)FROM customers;

### GROUP BY

The GROUP BY statement groups rows with the same values into summary rows. The statement is often used with aggregate functions. For example, the code below will display the average age for each name that appears in our customers table.

SELECT name, AVG(age)FROM customersGROUP BY name;

### HAVING

HAVING performs the same action as the WHERE clause. The difference is that HAVING is used for aggregate functions, whereas WHERE doesn’t work with them.

The below example would return the number of rows for each name, but only for names with more than 2 records.

SELECT COUNT(customer\_id), nameFROM customersGROUP BY nameHAVING COUNT(customer\_id) > 2;

### ORDER BY

ORDER BY sets the order of the returned results. The order will be ascending by default.

SELECT nameFROM customersORDER BY age;

### DESC

DESC will return the results in descending order.

SELECT nameFROM customersORDER BY age DESC;

### COMMIT

COMMIT is for saving every transaction to the database. A COMMIT statement will release any existing savepoints that may be in use and once the statement is issued, you cannot roll back the transaction.

DELETE FROM customersWHERE name = ‘Bob’;COMMIT

### ROLLBACK

ROLLBACK is used to undo transactions which are not saved to the database. This can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued. You can also rollback to a SAVEPOINT that has been created before.

### TRUNCATE

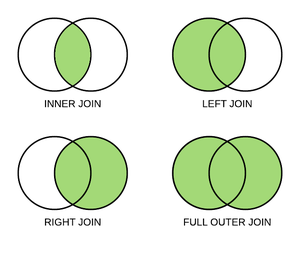
TRUNCATE TABLE removes all data entries from a table in a database, but keeps the table and structure in place. Similar to DELETE.

TRUNCATE TABLE customers;

### Join Clause

 A JOIN clause combines the records from two tables on the basis of common attributes. The different types of joins are as follows:

* **INNER JOIN**(OR JOIN) – Gives the records that have common attributes in both tables.
* **LEFT JOIN –**Gives all records from the left table and only the common records from the right table.
* **RIGHT JOIN –**Gives all records from the right table and only the common records from the left table.
* **FULL OUTER JOIN –**Gives all records when there is a common attribute in either the left or the right table.
* **CROSS JOIN –**Gives records of one table with all other records of another table.



* Unlike other types of joins, cross join does not include a join condition.
* SQLite does not directly support the RIGHT JOIN and FULL OUTER JOIN.

### ****INNER JOIN****

Inner join also represented as join which gives the records that have common attributes in both tables.

***Syntax:***

*SELECT columns*

*FROM table1*

*[INNER] JOIN table2*

*ON table1.column = table2.column;*

*INNER keyword is optional*

### ****LEFT JOIN****

Gives all records from the left table, and only the common records from the right table.

***Syntax:***

*SELECT columns*

*FROM table1*

*LEFT [OUTER] JOIN table2*

*ON table1.column = table2.column;*

*OUTER keyword is optional*

**CROSS JOIN**

It combines all records of one table with all other records of another table, that is, it creates a Cartesian product of records from the join tables.

***Syntax:***

*SELECT columns*

*FROM table1*

*CROSS JOIN table2;*