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 name

FROM 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 name

FROM customers;
```

SELECT INTO

SELECT INTO copies the specified data from one table into another.

```
SELECT * INTO customers

FROM customers_backup;
```

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_name

FROM customers;
```

FROM

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

```
SELECT name

FROM 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 name

FROM customers

WHERE 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 name

FROM customers

WHERE 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 name

FROM customers

WHERE name = ‘Bob’ OR age = 55;
```

BETWEEN

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

```
SELECT name

FROM customers

WHERE 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 name

FROM customers

WHERE 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 name

FROM customers

WHERE name IN (‘Bob’, ‘Fred’, ‘Harry’);
```

IS NULL

IS NULL will return only rows with a NULL value.

```
SELECT name

FROM customers

WHERE name IS NULL;
```

IS NOT NULL

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

```
SELECT name

FROM customers

WHERE 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

);
```

CREATE INDEX

CREATE INDEX generates an index for a table. Indexes are used to retrieve data from a database faster.

```
CREATE INDEX idx_name

ON customers (name);
```

CREATE VIEW

CREATE VIEW creates a virtual table based on the result set of an SQL statement. A view is like a regular table (and can be queried like one), but it is *not* saved as a permanent table in the database.

```
CREATE VIEW [Bob Customers] AS

SELECT name, age

FROM customers

WHERE name = 'Bob';
```

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 customers

SET age = 56

WHERE 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 customers

WHERE 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 customers

ADD surname varchar(255);

ALTER TABLE customers

DROP 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 customers

GROUP 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), name

FROM customers

GROUP BY name

HAVING COUNT(customer_id) > 2;
```

ORDER BY

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

```
SELECT name

FROM customers

ORDER BY age;
```

DESC

DESC will return the results in descending order.

```
SELECT name

FROM customers

ORDER BY age DESC;
```

OFFSET

The OFFSET statement works with ORDER BY and specifies the number of rows to skip before starting to return rows from the query.

```sql
SELECT name

FROM customers

ORDER BY age

OFFSET 10 ROWS;
```

FETCH

FETCH specifies the number of rows to return after the OFFSET clause has been processed. The OFFSET clause is mandatory, while the FETCH clause is optional.

```sql
SELECT name

FROM customers

ORDER BY age

OFFSET 10 ROWS

FETCH NEXT 10 ROWS ONLY;
```

JOINS (INNER, LEFT, RIGHT, FULL)

A JOIN clause is used to combine rows from two or more tables. The four types of JOIN are INNER, LEFT, RIGHT and FULL.

INNER JOIN

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

```sql
SELECT name

FROM customers

INNER JOIN orders

ON customers.customer_id = orders.customer_id;
```

LEFT JOIN

LEFT JOIN selects records from the left table that match records in the right table. In the below example the left table is `customers`.

```sql
SELECT name

FROM customers

LEFT JOIN orders

ON customers.customer_id = orders.customer_id;
```

RIGHT JOIN

RIGHT JOIN selects records from the right table that match records in the left table. In the below example the right table is `orders`.

```sql
SELECT name
```

```
FROM customers

RIGHT JOIN orders

ON customers.customer_id = orders.customer_id;
```

FULL JOIN

FULL JOIN selects records that have a match in the left or right table. Think of it as the “OR” JOIN compared with the “AND” JOIN (INNER JOIN).

```
SELECT name

FROM customers

FULL OUTER JOIN orders

ON customers.customer_id = orders.customer_id;
```

EXISTS

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

```
SELECT name

FROM customers

WHERE EXISTS

(SELECT order FROM ORDERS WHERE customer_id = 1);
```

GRANT

GRANT gives a particular user access to database objects such as tables, views or the database itself. The below example would give SELECT and UPDATE access on the customers table to a user named “usr_bob”.

```
GRANT SELECT, UPDATE ON customers TO usr_bob;
```

REVOKE

REVOKE removes a user's permissions for a particular database object.

```
REVOKE SELECT, UPDATE ON customers FROM usr_bob;
```

SAVEPOINT

SAVEPOINT allows you to identify a point in a transaction to which you can later roll back. Similar to creating a backup.

```
SAVEPOINT SAVEPOINT_NAME;
```

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 customers

WHERE 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.

```
ROLLBACK TO SAVEPOINT_NAME;
```

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;
```

UNION

UNION combines multiple result-sets using two or more SELECT statements and eliminates duplicate rows.

SELECT name FROM customersUNIONSELECT name FROM orders;

UNION ALL

UNION ALL combines multiple result-sets using two or more SELECT statements and keeps duplicate rows.

```
SELECT name FROM customers

UNION

SELECT name FROM orders;
```