February 17, 2021

SQL Commands: The Complete List (w/ Examples)

To get a data job, you are going to need to learn SQL. The common SQL commands and operators discussed in this post are a great reference. If you're looking for more details and SQL resources, check out our Complete guide to SQL.

Below is a comprehensive list of SQL commands, organized by the top-level of each (e.g. SELECT TOP is within the SELECT category).

If you're on a journey to learn SQL and you've been frustrated by the lack of structure or the dull curriculum, then you may like Dataquest's interactive SQL skill path. Try it free.

Here are a few SQL courses:

1. Introduction to SQL and Databases
2. Filtering and Sorting Data in SQL
3. Summarizing Data in SQL
4. Combining Tables in SQL
5. SQL Subqueries

List of SQL Commands

SELECT

SELECT is probably the most commonly-used SQL statement.

```
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)

and returns a single result.

COUNT

COUNT returns the number of rows that match the specified c ia. In the code below, we're using *, so the total row count f stomers 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.

```
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.

```
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.

```
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`.

```
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`.

```
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;
```

We hope this page serves as a helpful quick-reference guide to SQL commands. But if you really want to learn your SQL skills,