DATAQUEST

SQL Cheat Sheet: Fundamentals
Performing calculations with SQL

Performing a single calculation:

```
SELECT 1320+17;
```

Performing multiple calculations:

```
SELECT 1320+17, 1340-3, 7*191, 8022/6;
```

Performing calculations with multiple numbers:

```
SELECT 1*2*3, 1+2+3;
```

Renaming results:

```
SELECT 2*3 AS mult, 1+2+3 AS nice_sum;
```

Selecting tables, columns, and rows:

Remember: The order of clauses matters in SQL. SQL uses the following order of precedence: FROM, SELECT, LIMIT.

Display the whole table:

```
SELECT *
  FROM table_name;
```

Select specific columns from a table:

```
SELECT column_name_1, column_name_2
  FROM table_name;
```

Display the first 10 rows on a table:

```
SELECT *
  FROM table_name;
 LIMIT 10;
```

Adding comments to your SQL queries

Adding single-line comments:

```
-- First comment
SELECT column_1, column_2, column_3 -- Second comment
  FROM table_name; -- Third comment
```

Adding block comments:

```
/*
This comment
spans over
multiple lines
 */
SELECT column_1, column_2, column_3
  FROM table_name;
```

SQL Intermediate:
Joins & Complex Queries

Many of these examples use table and column names from the real SQL databases that learners work with in our interactive SQL courses. For more information, sign up for a free account and try one out!

Joining data in SQL:

Joining tables with INNER JOIN:

```
SELECT column_name_1, column_name_2 FROM table_name_1
INNER JOIN table_name_2 ON table_name_1.column_name_1
          = table_name_2.column_name_1;
```

Joining tables using a LEFT JOIN:

```
SELECT * FROM facts
LEFT JOIN cities ON cities.facts_id = facts.id;
```

Joining tables using a RIGHT JOIN:

```
SELECT f.name country, c.name city
FROM cities c
RIGHT JOIN facts f ON f.id = c.facts;
```

Joining tables using a FULL OUTER JOIN:

```
SELECT f.name country, c.name city
FROM cities c
FULL OUTER JOIN facts f ON f.id = c.facts_id;
```

Sorting a column without specifying a column name:

```
SELECT name, migration_rate FROM FACTS
ORDER BY 2 desc; -- 2 refers to migration_rate column
```

Using a join within a subquery, with a limit:

```
SELECT c.name capital_city, f.name country
FROM facts f
INNER JOIN (
        SELECT * FROM cities
                WHERE capital = 1
                ) c ON c.facts_id = f.id
INNER 10
```

Joining data from more than two tables:

```
SELECT [column_names] FROM [table_name_one]
   [join_type] JOIN [table_name_two] ON [join_constraint]
      [join_type] JOIN [table_name_three] ON [join_constraint]
      ...
      ...
      ...
      [join_type] JOIN [table_name_three] ON [join_constraint]
```

Other common SQL operations:

Combining columns into a single column:

```
SELECT
          album_id,
          artist_id,
          "album id is " || album_id col_1,
          "artist id is " || artist_id col2,
          album_id || artist_id col3
FROM album LIMIT 3;
```

Matching part of a string:

```
SELECT
     first_name,
     last_name,
     phone
FROM customer
WHERE first_name LIKE "%Jen%";
```

Using if/then logic in SQL with CASE:

```
CASE
     WHEN [comparison_1] THEN [value_1]
     WHEN [comparison_2] THEN [value_2]
     ELSE [value_3]
     END
AS [new_column_name]
```

Using the WITH clause:

```
WITH track_info AS
(
     SELECT
          t.name,
          ar.name artist,
          al.title album_name,
     FROM track t
     INNER JOIN album al ON al.album_id = t.album_id
     INNER JOIN artist ar ON ar.artist_id = al.artist_id
)
SELECT * FROM track_info
WHERE album_name = "Jagged Little Pill";
```

Creating a view:

```
CREATE VIEW chinook.customer_2 AS
SELECT * FROM chinook.customer;
```

Dropping a view

```
DROP VIEW chinook.customer_2;
```

Selecting rows that occur in one or more SELECT statements:

```
[select_statement_one]
UNION
[select_statement_two];
```

Selecting rows that occur in both SELECT statements:

```
SELECT * from customer_usa
INTERSECT
SELECT * from customer_gt_90_dollars;
```

Selecting rows that occur in the first SELECT statement but not the second SELECT statement:

```
SELECT * from customer_usa
EXCEPT
SELECT * from customer_gt_90_dollars;
```

Chaining WITH statements:

```
WITH
usa AS
     (
     SELECT * FROM customer
     WHERE country = "USA"
     ),
last_name_g AS
     (
     SELECT * FROM usa
     WHERE last_name LIKE "G%"
     ),
state_ca AS
     (
     SELECT * FROM last_name_g
     WHERE state = "CA"
     )
SELECT
     first_name,
     last_name,
     country,
     state
FROM state_ca
```

Important Concepts and Resources:
Reserved words

Reserved words are words that cannot be used as identifiers (such as variable names or function names) in a programming language, because they have a specific meaning in the language itself. Here is a list of reserved words in SQL.