BigQuery SQL Cheat Sheet

Basic SQL Commands

1. **CREATE TABLE**

Syntax:
sql
code:
```
CREATE TABLE `project_id.dataset_id.table_name` (
  column_name1 DATA_TYPE,
  column_name2 DATA_TYPE,
  ...
);
```

-

Example:
sql
code:
```
CREATE TABLE `my_project.my_dataset.daily_activity_1` (
  user_id STRING,
  activity_date DATE,
  steps INTEGER,
  calories FLOAT64
);
```

-

2. **DROP TABLE**

Syntax:
sql
code:
```
DROP TABLE `project_id.dataset_id.table_name`;
```

-

3. **SELECT**

Syntax:
sql

code:
```
SELECT column1, column2, ...
FROM `project_id.dataset_id.table_name`
WHERE condition;
```

-

Example:
sql
code:
```
SELECT user_id, steps
FROM `my_project.my_dataset.daily_activity_1`
WHERE activity_date = '2024-08-26';
```

-

4. **INSERT INTO**

Syntax:
sql
code:
```
INSERT INTO `project_id.dataset_id.table_name` (column1, column2, ...)
VALUES (value1, value2, ...);
```

-

Example:
sql
code:
```
INSERT INTO `my_project.my_dataset.daily_activity_1` (user_id,
activity_date, steps, calories)
VALUES ('user123', '2024-08-26', 10000, 250.5);
```

-

5. **UPDATE**

Syntax:
sql
code:
```
UPDATE `project_id.dataset_id.table_name`
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

-

Example:
sql
code:
```
UPDATE `my_project.my_dataset.daily_activity_1`
SET steps = 11000
WHERE user_id = 'user123' AND activity_date = '2024-08-26';
```

-
6. **DELETE**

Syntax:
sql
code:
```
DELETE FROM `project_id.dataset_id.table_name`
WHERE condition;
```

-

Example:
sql
code:
```
DELETE FROM `my_project.my_dataset.daily_activity_1`
WHERE user_id = 'user123' AND activity_date = '2024-08-26';
```

-
7. **ALTER TABLE**

Syntax:
sql
code:
```
ALTER TABLE `project_id.dataset_id.table_name`
ADD COLUMN new_column_name DATA_TYPE;
```

-

Example:
sql
code:
```
ALTER TABLE `my_project.my_dataset.daily_activity_1`
ADD COLUMN new_column STRING;
```

-

Common SQL Functions

1. **Aggregate Functions**

COUNT():
sql
code:
```
SELECT COUNT(*) FROM `project_id.dataset_id.table_name`;
```

-

SUM():
sql
code:
```
SELECT SUM(column_name) FROM `project_id.dataset_id.table_name`;
```

-

AVG():
sql
code:
```
SELECT AVG(column_name) FROM `project_id.dataset_id.table_name`;
```

-

MAX():
sql
code:
```
SELECT MAX(column_name) FROM `project_id.dataset_id.table_name`;
```

-

MIN():
sql
code:
```
SELECT MIN(column_name) FROM `project_id.dataset_id.table_name`;
```

-

2. **String Functions**

CONCAT():
sql
code:
```
SELECT CONCAT(column1, column2) AS combined_column FROM
`project_id.dataset_id.table_name`;
```

-

UPPER():
sql
code:
```
SELECT UPPER(column_name) FROM `project_id.dataset_id.table_name`;
```

-

LOWER():
sql
code:
```
SELECT LOWER(column_name) FROM `project_id.dataset_id.table_name`;
```

-

SUBSTRING():
sql
code:
```
SELECT SUBSTRING(column_name, start_position, length) FROM
`project_id.dataset_id.table_name`;
```

-

3. **Date Functions**

CURRENT_DATE():
sql
code:
```
SELECT CURRENT_DATE();
```

-

DATE():
sql
code:
```
SELECT DATE('2024-08-26');
```

-

EXTRACT():
sql
code:
```
SELECT EXTRACT(YEAR FROM date_column) FROM
`project_id.dataset_id.table_name`;
```

-

4. **Conditional Functions**

IF():
sql
code:
```
SELECT IF(condition, value_if_true, value_if_false) FROM
`project_id.dataset_id.table_name`;
```

-

COALESCE():
sql
code:
```
SELECT COALESCE(column1, column2, 'default_value') FROM
`project_id.dataset_id.table_name`;
```

-

Advanced SQL Commands

1. **JOIN**

Syntax:
sql
code:
```
SELECT a.column1, b.column2
FROM `project_id.dataset_id.table_a` a
JOIN `project_id.dataset_id.table_b` b
ON a.common_column = b.common_column;
```

-

2. **GROUP BY**

Syntax:
sql
code:
```
SELECT column1, COUNT(*)
FROM `project_id.dataset_id.table_name`
GROUP BY column1;
```

-

3. **ORDER BY**

Syntax:
sql
code:

```
SELECT column1, column2
FROM `project_id.dataset_id.table_name`
ORDER BY column1 ASC;
```

4. **LIMIT**

Syntax:
sql
code:

```
SELECT *
FROM `project_id.dataset_id.table_name`
LIMIT 10;
```

BigQuery-Specific Functions

1. **ARRAY_AGG()**

Syntax:
sql
code:

```
SELECT ARRAY_AGG(column_name) FROM `project_id.dataset_id.table_name`;
```

2. **STRUCT()**

Syntax:
sql
code:

```
SELECT STRUCT(column1 AS field1, column2 AS field2) FROM
`project_id.dataset_id.table_name`;
```

3. **GENERATE_ARRAY()**

Syntax:
sql
code:

```
SELECT GENERATE_ARRAY(1, 10);
```

-

Basic Syntax Rules

1. **Semicolon (`;`)**
   - End each SQL statement with a semicolon.
2. **Identifiers**
   - Use backticks (`) to enclose project, dataset, and table names, especially if they contain special characters or reserved words.
3. **String Literals**
   - Enclose string literals in single quotes (').
4. **Comments**
   - Use `--` for single-line comments and `/* ... */` for multi-line comments.

Types of sql joins:

1. INNER JOIN

- **Purpose**: Returns rows that have matching values in both tables.
- **Result**: Only rows with matching keys in both tables are included.

Example:
sql
code:

```
SELECT a.*, b.*
FROM table1 a
INNER JOIN table2 b
ON a.id = b.id;
```

-

2. LEFT JOIN (or LEFT OUTER JOIN)

- **Purpose**: Returns all rows from the left table and the matched rows from the right table. If no match is found, NULL values are returned for columns from the right table.
- **Result**: Includes all rows from the left table, with matched rows from the right table.

Example:
sql
code:

```
SELECT a.*, b.*
FROM table1 a
```

```
LEFT JOIN table2 b
ON a.id = b.id;
```

-

3. RIGHT JOIN (or RIGHT OUTER JOIN)

- **Purpose**: Returns all rows from the right table and the matched rows from the left table. If no match is found, NULL values are returned for columns from the left table.
- **Result**: Includes all rows from the right table, with matched rows from the left table.

Example:
sql
code:
```
SELECT a.*, b.*
FROM table1 a
RIGHT JOIN table2 b
ON a.id = b.id;
```

-

4. FULL JOIN (or FULL OUTER JOIN)

- **Purpose**: Returns all rows when there is a match in either the left or right table. Non-matching rows will have NULL values for columns from the table without a match.
- **Result**: Includes all rows from both tables, with NULLs where there are no matches.

Example:
sql
code:
```
SELECT a.*, b.*
FROM table1 a
FULL JOIN table2 b
ON a.id = b.id;
```

-

5. CROSS JOIN

- **Purpose**: Returns the Cartesian product of both tables. Every row from the first table is combined with every row from the second table.
- **Result**: Includes all possible combinations of rows from both tables.

Example:
sql
code:

```
SELECT a.*, b.*
FROM table1 a
CROSS JOIN table2 b;
```

-

6. SELF JOIN

- **Purpose**: Joins a table with itself. Useful for comparing rows within the same table.
- **Result**: Includes rows from the same table with different aliases.

Example:
sql
code:

```
SELECT a.*, b.*
FROM table1 a
INNER JOIN table1 b
ON a.id = b.related_id;
```

-

These joins allow you to combine data from multiple tables in various ways to meet your querying needs.