iykra Data Fellowship

IYKRA
Data Fellowship Program

BigQuery SQL

by Fariz Wakan

April 22th, 2024

Data Fellowship

Intro to SQL

What is database?

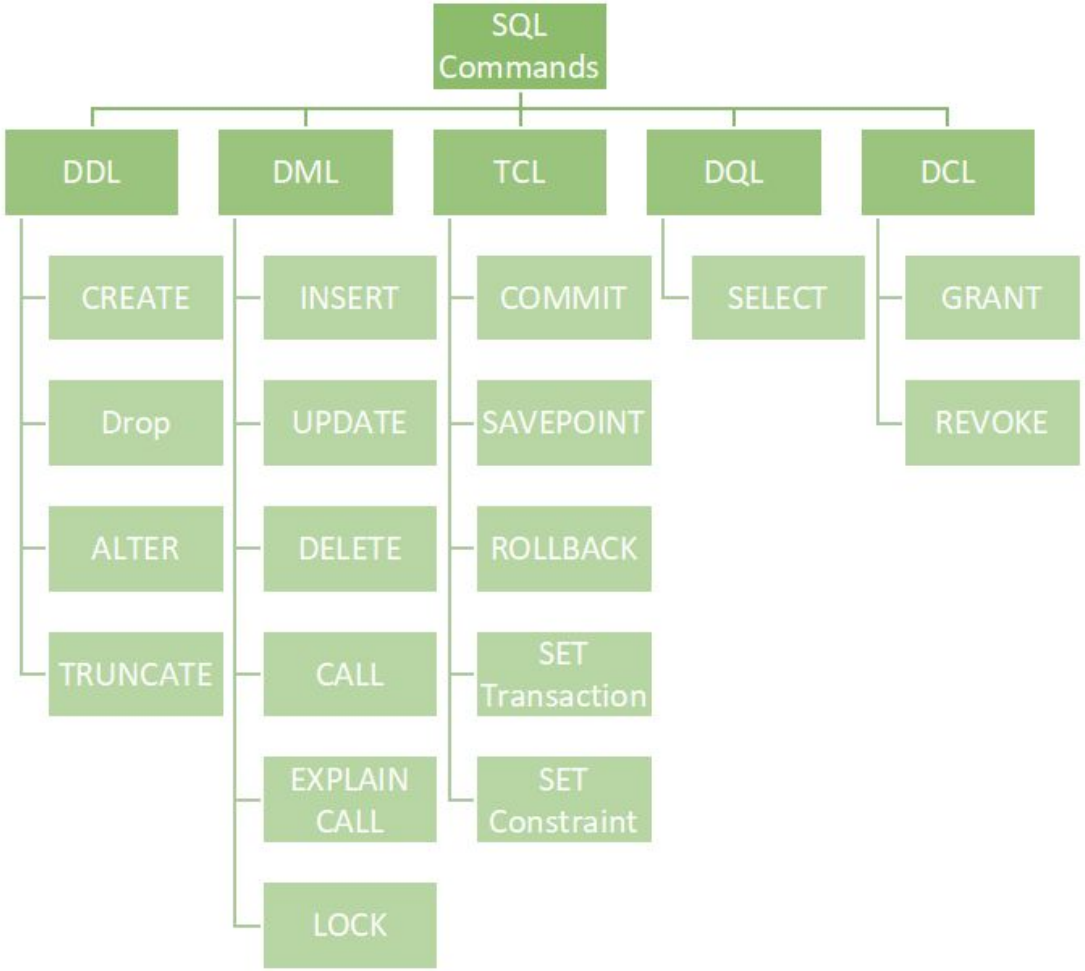
- A database is an **organized collection of data stored in a computer system** and usually **controlled by a database management system** (DBMS).
- The data in common databases is modeled in tables (columns & rows), making querying and processing efficient.
- Structured query language (SQL) is commonly used for data querying and writing.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you **access and manipulate databases**
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

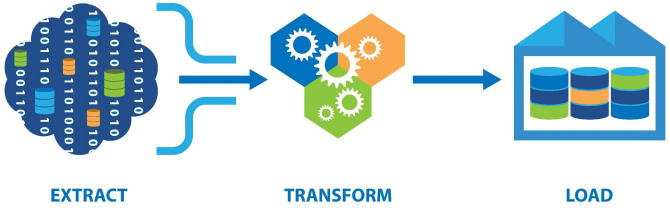
What can SQL do?

- SQL can *execute queries* against a database
- SQL can *retrieve data* from a database
- SQL can *insert records* in a database
- SQL can *update records* in a database
- SQL can *delete records* from a database
- SQL can *create new databases*
- SQL can *create new tables* in a database
- SQL can *create stored procedures* in a database
- SQL can *create views* in a database
- SQL can *set permissions* on tables, procedures, and views
- *etc ..*


SQL Commands
DDL
DML
TCL
DQL
DCL
CREATE
Drop
ALTER
TRUNCATE
INSERT
UPDATE
DELETE
CALL
EXPLAIN CALL
LOCK
COMMIT
SAVEPOINT
ROLLBACK
SET Transaction
SET Constraint
SELECT
GRANT
REVOKE

SQL in Data Engineering

SQL plays a crucial role in data engineering, particularly in the processes of data ingestion (from various sources) and data transformation (shaping and preparing data for analysis or storage)



Data Fellowship

SQL in BigQuery

...

Introduction to SQL in BigQu

cloud.google.com/bigquery/docs/introduction-sql

Google Cloud | Technology areas | Cross-product tools | Related sites | Search | English | Console

BigQuery | Guides | Reference | Samples | Resources | Contact Us

- Quickstarts
- Explore BigQuery tools

Migrate
- Overview
- Migrate a data warehouse
- Migration guides

Design
- Organize resources
- Understand editions
- Datasets
- Tables
- Views
- Routines
- Connections
- Search indexes
- Vector indexes

Load, Transform, Export
- Load data
- Transform data
- Export data

Analyze
- Introduction

BigQuery > Documentation > Guides

Was this helpful?

Introduction to SQL in BigQuery

Send feedback

GoogleSQL is the new name for Google Standard SQL! New name, same great SQL dialect.

This document provides an overview of supported statements and SQL dialects in BigQuery.

Overview

GoogleSQL is an ANSI compliant Structured Query Language (SQL) which includes the following types of supported statements:

- Query statements, also known as Data Query Language (DQL) statements, are the primary method to analyze data in BigQuery. They scan one or more tables or expressions and return the computed result rows.
- Procedural language statements are procedural extensions to GoogleSQL that allow you to execute multiple SQL statements in one request. Procedural statements can use variables and control-flow statements, and can have side effects.
- Data Definition Language (DDL) statements let you create and modify database objects such as tables, views, functions, and row-level access policies.

On this page
- Overview
- BigQuery SQL dialects
  - Changing from the default dialect
- What's next

Basic SQL

SELECT, used to retrieve data from one or more tables.

```
SELECT column1, column2, …
FROM table_name
WHERE condition;
```

INSERT, used to add new records (rows) to a table.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Basic SQL

UPDATE, used to modify existing records in a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, …
WHERE condition;
```

DELETE, used to remove records from a table.

```
DELETE FROM table_name
WHERE condition;
```

WHERE

- The **WHERE** clause is used to filter records
- It is used to extract only those records that fulfill a specified condition
  - When the output condition is **TRUE**

```
SELECT column1, column2, …
FROM table_name
WHERE condition;
```

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT age, country
FROM Customers
WHERE country = 'USA';
```

age	country
31	USA
22	USA

WHERE: Operators

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

WHERE: AND, OR, NOT

- You may want to have **more than one condition**
  - The **AND** operator displays a record if *all the conditions are TRUE*

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND CustomerName LIKE 'G%';
```

  - The **OR** operator displays a record if *any of the conditions are TRUE*

```
SELECT * FROM Customers
WHERE Country = 'Germany' OR Country = 'Spain';
```

WHERE: AND, OR, NOT

- You may want to have **more than one condition**
  - The **NOT** operator displays a record if it *not fulfill the conditions (opposite)*

```
SELECT * FROM Customers
WHERE NOT Country = 'Spain';
```

SELECT DISTINCT

The **SELECT DISTINCT** statement is used to return only distinct (different) values.

```
SELECT DISTINCT column1, column2, …
FROM table_name;
```

Aggregate Functions

- The **COUNT()** function returns the number of rows that matches a specified criterion

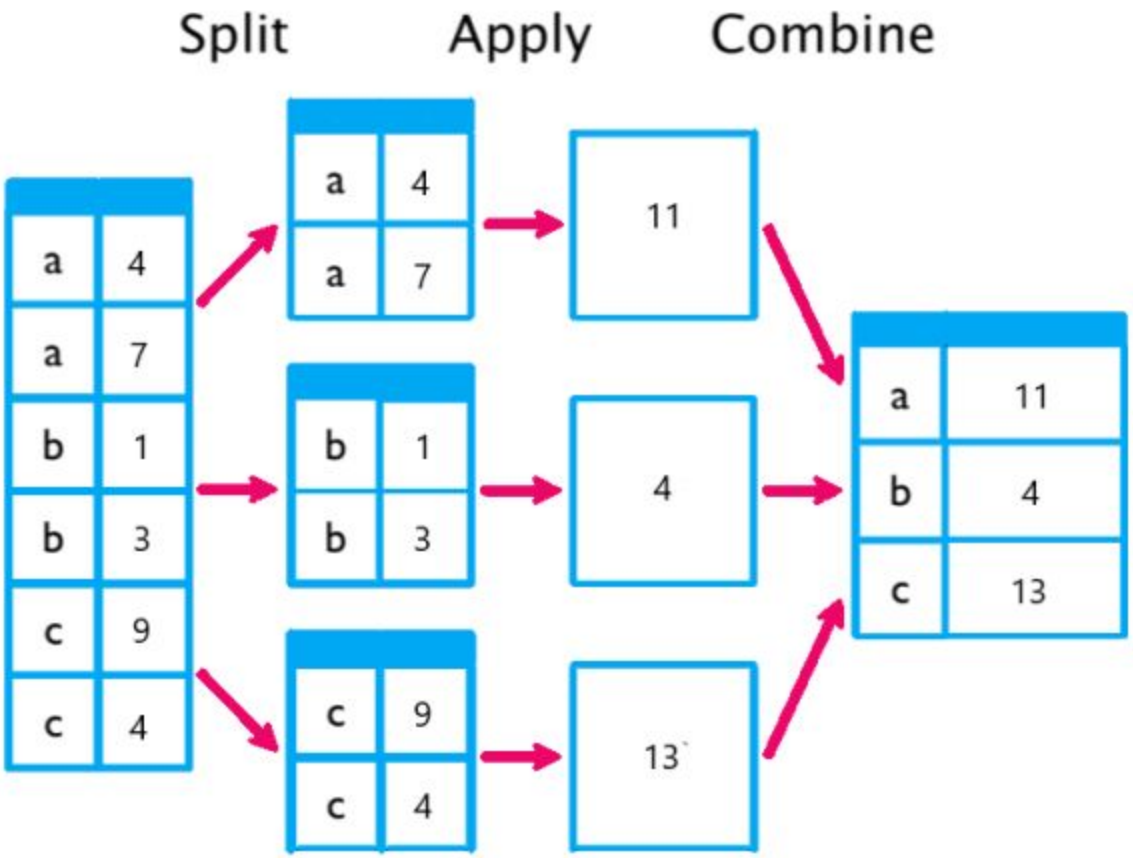
```
SELECT COUNT(*)
FROM Products;
```

- The **SUM()** function returns the total sum of a numeric column
- The **AVG()** function returns the average value of a numeric column
- The **MIN()** function returns the smallest value of the selected column
- The **MAX()** function returns the largest value of the selected column

GROUP BY

- The GROUP BY statement groups rows that have the same values into summary rows, like "*find the number of customers in each country*".
- The GROUP BY statement is often used with aggregate functions to group the result-set by one or more columns.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
```


Split
Apply
Combine
a 4
a 7
b 1
b 3
c 9
c 4
a 4
a 7
b 1
b 3
c 9
c 4
11
4
13
a 11
b 4
c 13

JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them. Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**, returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**, returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**, returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**, returns all records when there is a match in either left or right table

SQL INNER JOIN

```
SELECT
     customer_id,
     first_name,
     amount
FROM Customers c
INNER JOIN Orders o
     ON
          c.customer_id = o.customer;
```

Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
3	David	500
5	Betty	800

SQL LEFT JOIN

```
SELECT
     customer_id,
     first_name,
     amount
FROM Customers c
LEFT JOIN Orders o
     ON
          c.customer_id = o.customer;
```

Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
1	John	
2	Robert	
3	David	500
4	John	
5	Betty	800

SQL RIGHT JOIN

```
SELECT
     customer_id,
     first_name,
     amount
FROM Orders o
LEFT JOIN Customers c
     ON
          c.customer_id = o.customer;
```

Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
3	David	500
5	Betty	800
		200
		300
		150

SQL FULL OUTER JOIN

```
SELECT
     customer_id,
     first_name,
     amount
FROM Customers c
FULL JOIN Orders o
     ON
          c.customer_id = o.customer;
```

Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
		200
3	David	500
		300
5	Betty	800
		150
2	Robert	
4	John	

CTE

A **common table expression**, or CTE, is a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statement. Each SQL CTE is like a **named query**, whose result is stored in a virtual table (a CTE) to be referenced later in the main query.

```
WITH my_cte AS (
    SELECT column1, column2, column3
    FROM table_name
    WHERE conditions
)
SELECT column1, column2, column3
FROM my_cte
WHERE conditions;
```

Window Functions

Window functions applies aggregate and ranking functions over a particular window (set of rows). OVER clause is used with window functions to define that window. OVER clause does two things :

- Partitions rows into form set of rows. (PARTITION BY clause is used)
- Orders rows within those partitions into a particular order. (ORDER BY clause is used)

```
SELECT
    column1,
    window_function(column2) OVER(
        [PARTITION BY column1]
        [ORDER BY column3]
    ) AS column4
FROM table_name;
```

JULIA EVANS
@b0rk

SQL queries run in this order

FROM + JOIN

↓

WHERE

↓

GROUP BY

↓

HAVING

↓

SELECT (window functions happen here!)

↓

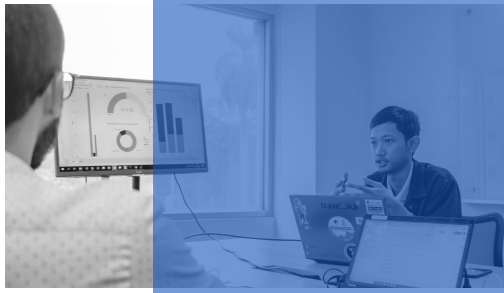
ORDER BY

↓

LIMIT

References

- https://cloud.google.com/bigquery/docs
- https://www.w3schools.com/sql/
- https://www.programiz.com/sql/
- https://www.geeksforgeeks.org/



iykra Data Fellowship

Thank you!