

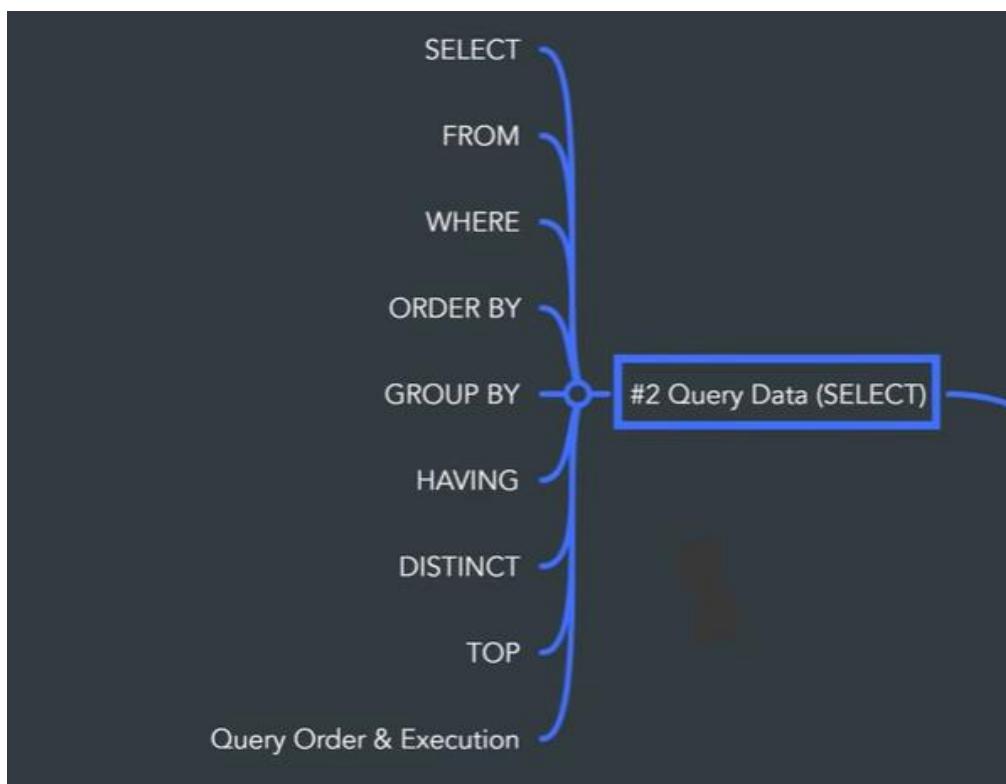
SQL notes

Agenda:



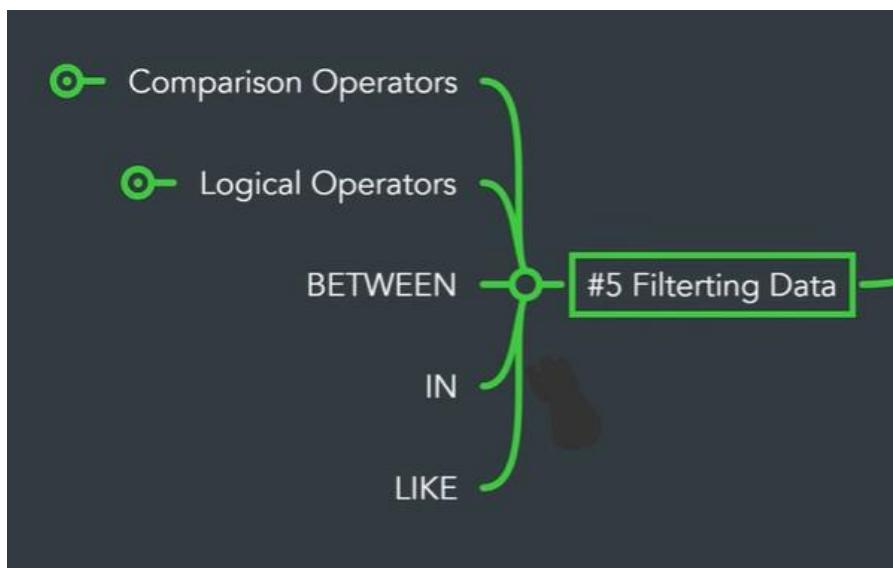
Detailed agenda:

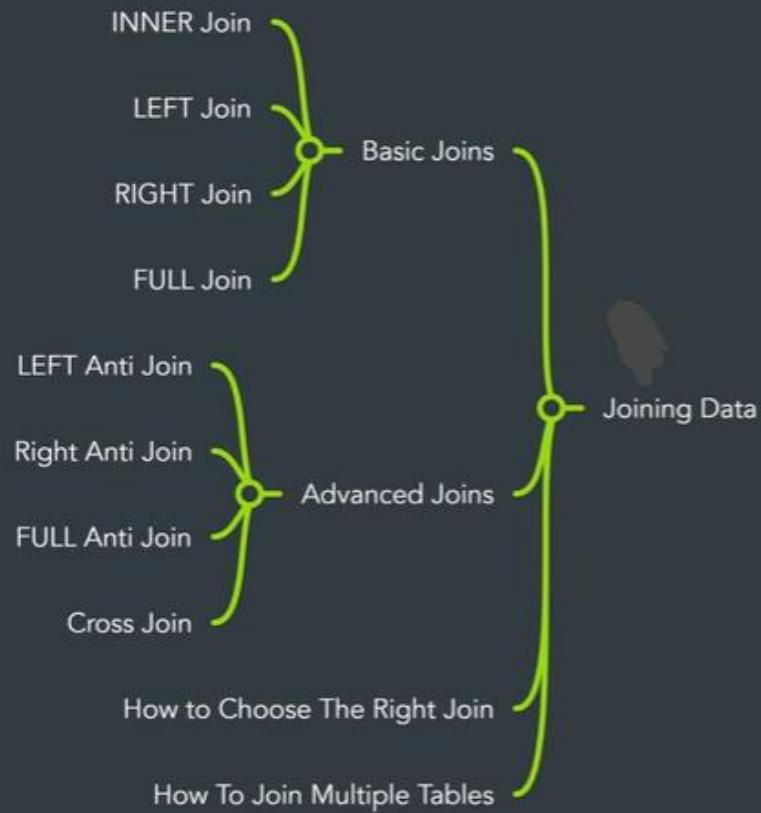
Basic level-

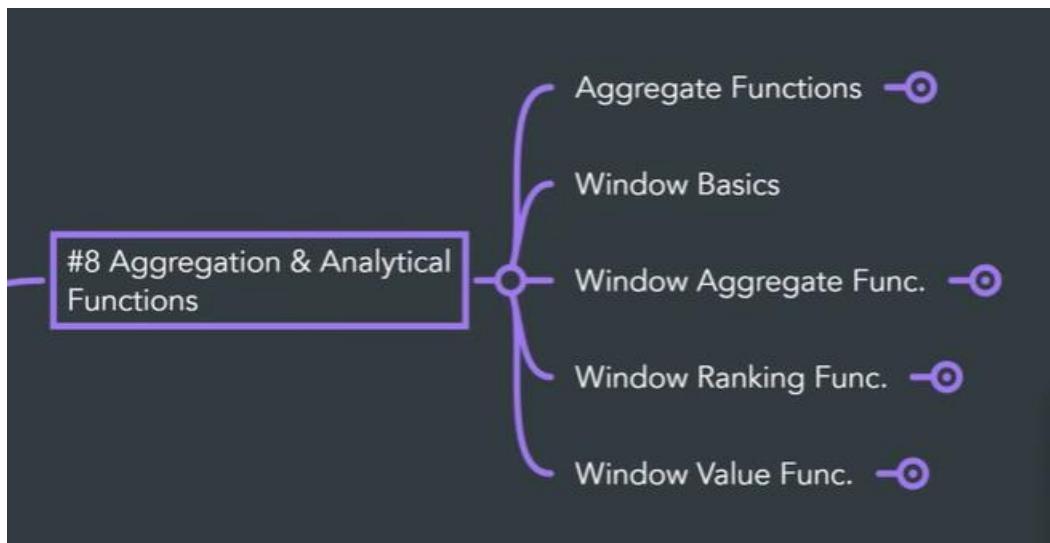
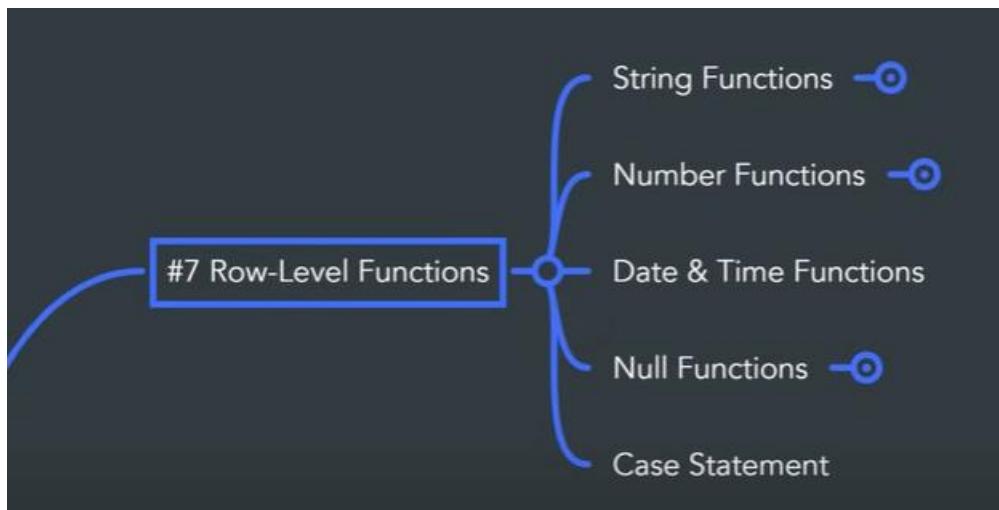




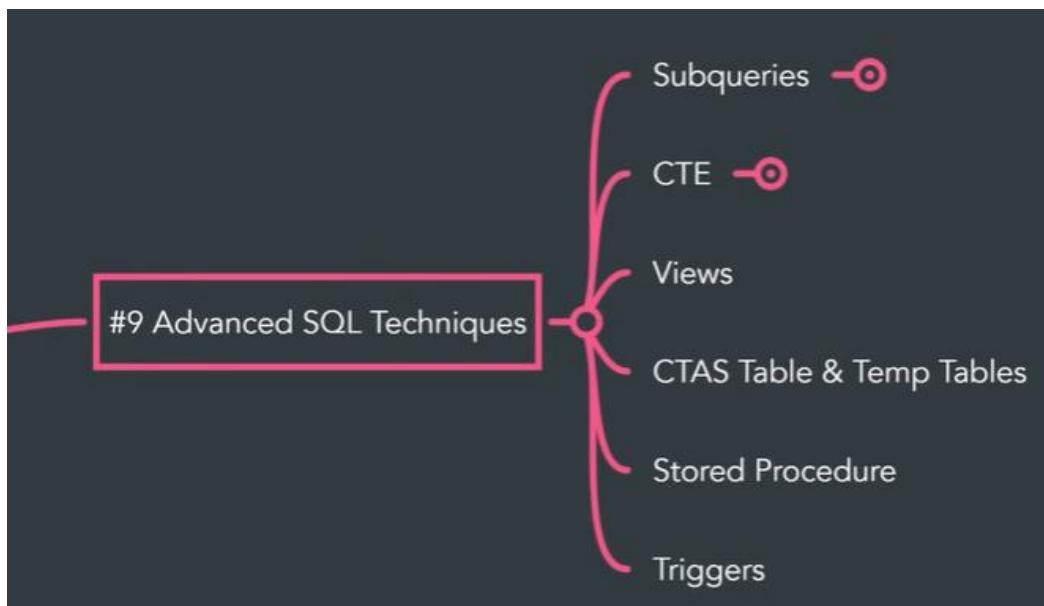
Intermediate level-

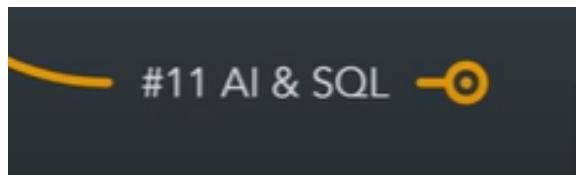






Advanced SQL-

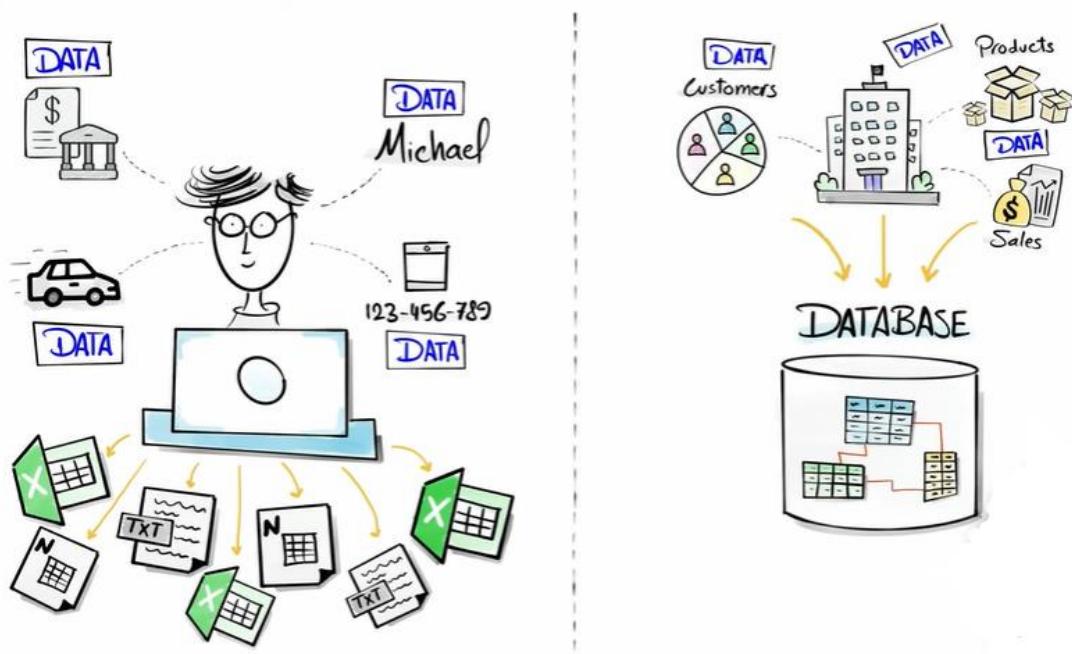




Introduction

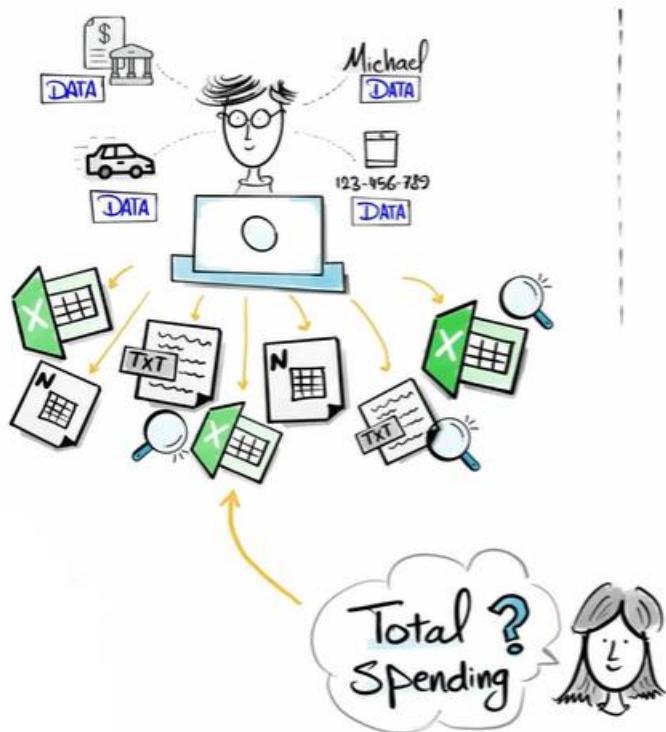
WHAT IS SQL ?

Everything generates data, everywhere we have data.



Where to store the data? - excel/text files are not a good option to store data.

Here is where Database comes into picture. Database is like a huge container which stores data in an organized manner for easy access and management of data.



With conventional excel and text files, it would be very difficult to understand data and get answers.



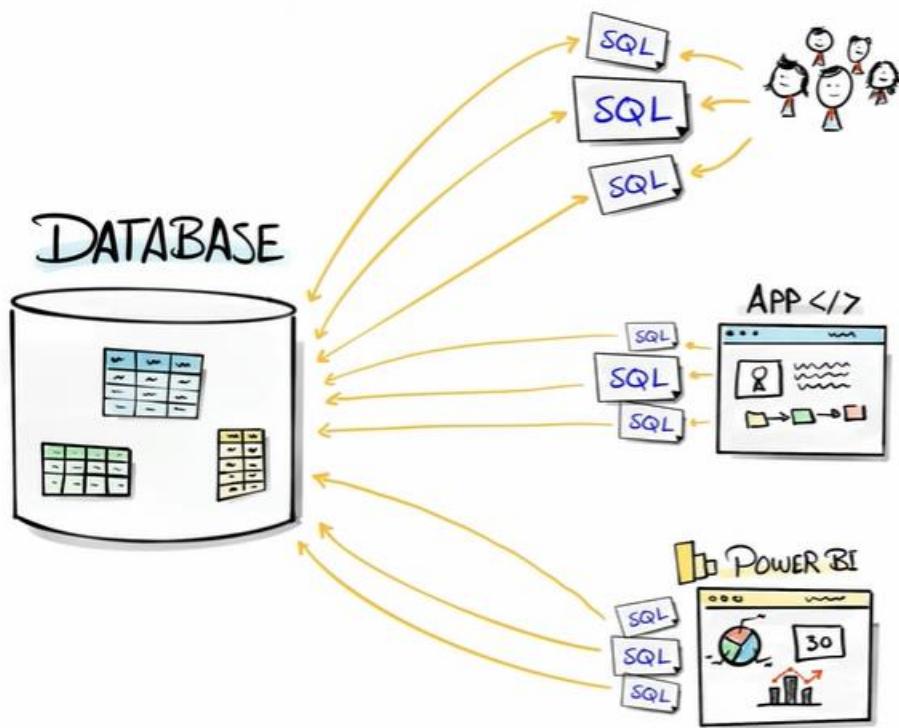
When data stores in a database, we can speak with the database about our query to get results.

How can we speak with database – using SQL – Structured Query language.

SQL Structured Query language is a programming language used to communicate with database.

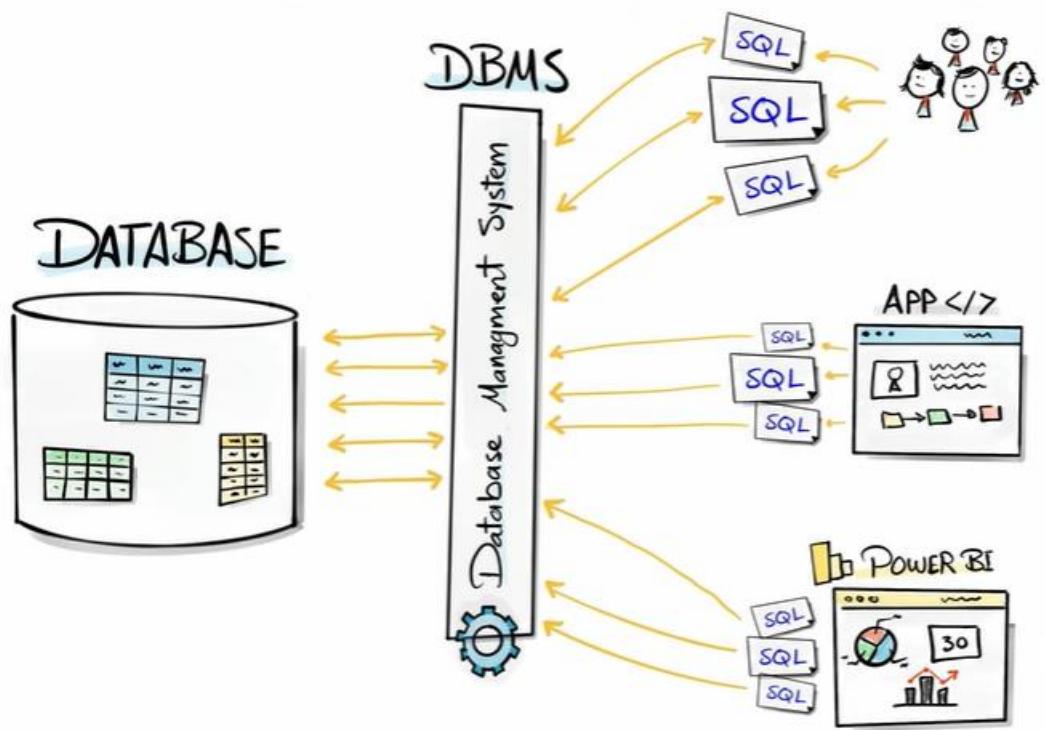
*Conventional excel/text files will not handle huge amounts of data, so we use Database which will handle huge amounts of data.

*Also, its safe and secure to store critical information in database than storing in Excel/text files and we can control who can have access.

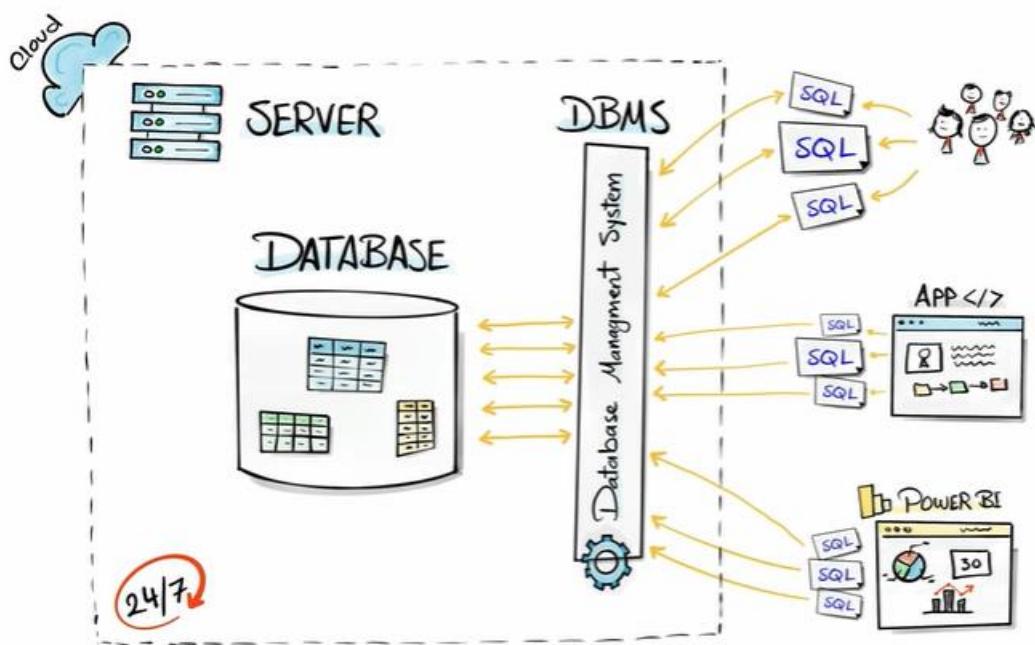


A database can have many requests from users, application and then Visualization tools.

In-order to manage all these requests we need a management system that will manage all the incoming requests. – A database Management system



Database Management system is a software that manages all the different requests to our database, and makes priority which query to be executed first and also checks the security.



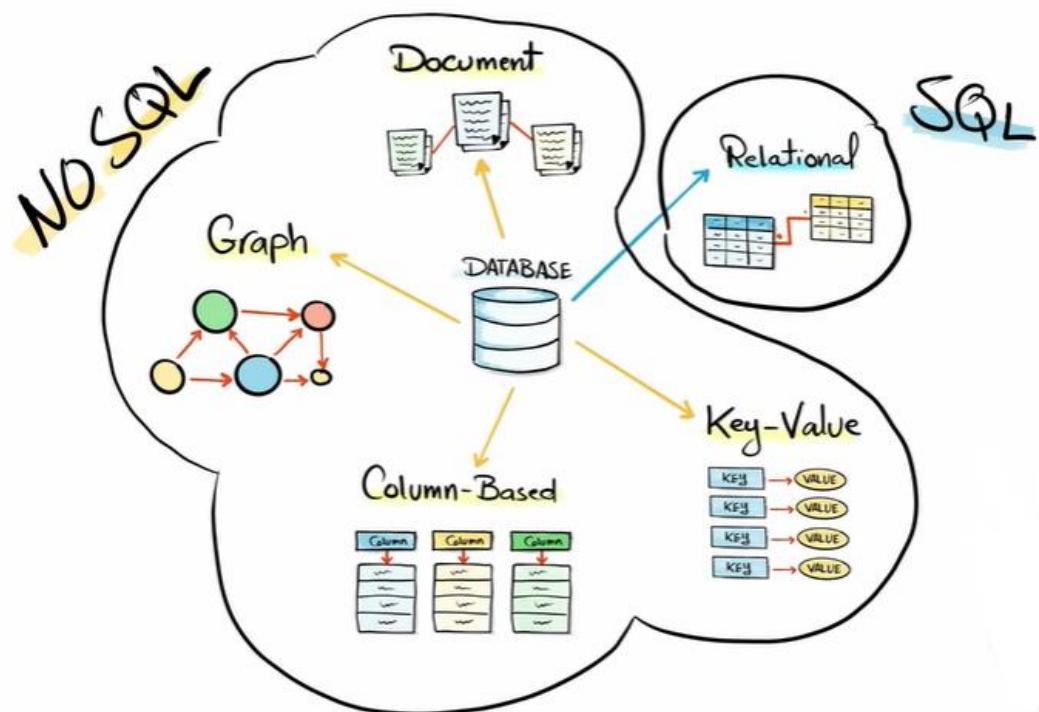
Database
Stores Data

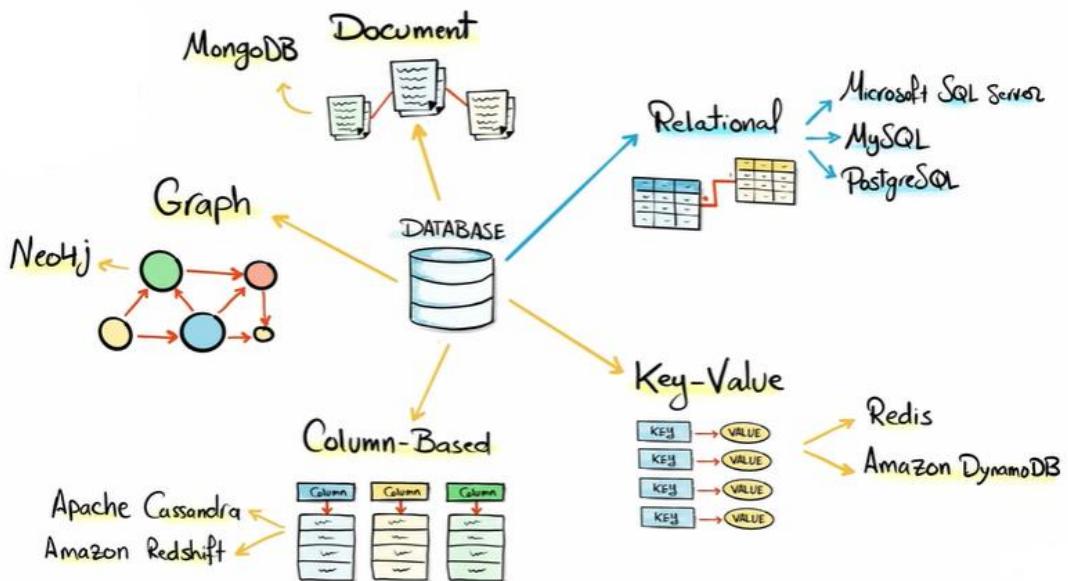
SQL
Speak to Database

DBMS
Manages Database

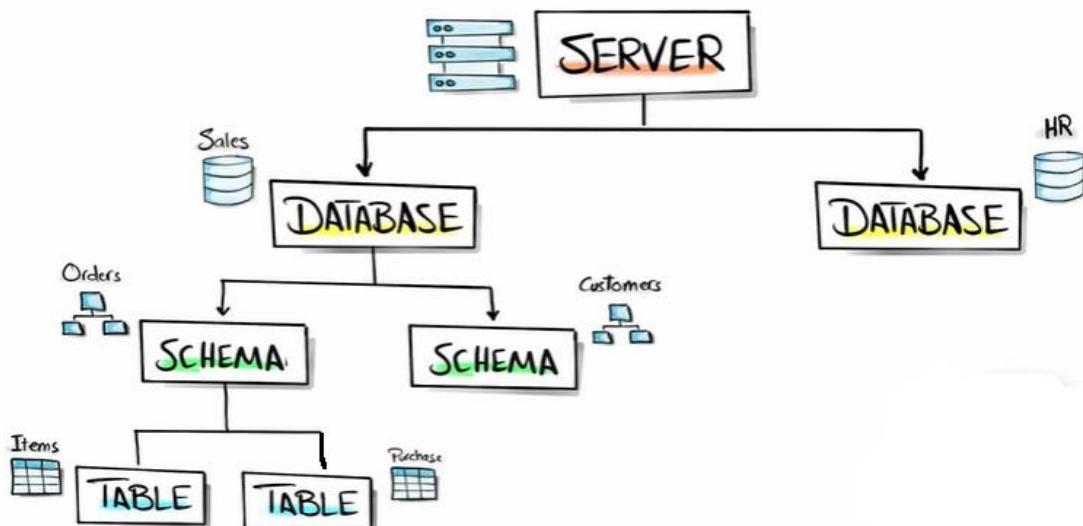
Server
Where Database lives

Types of Databases





Database Structure



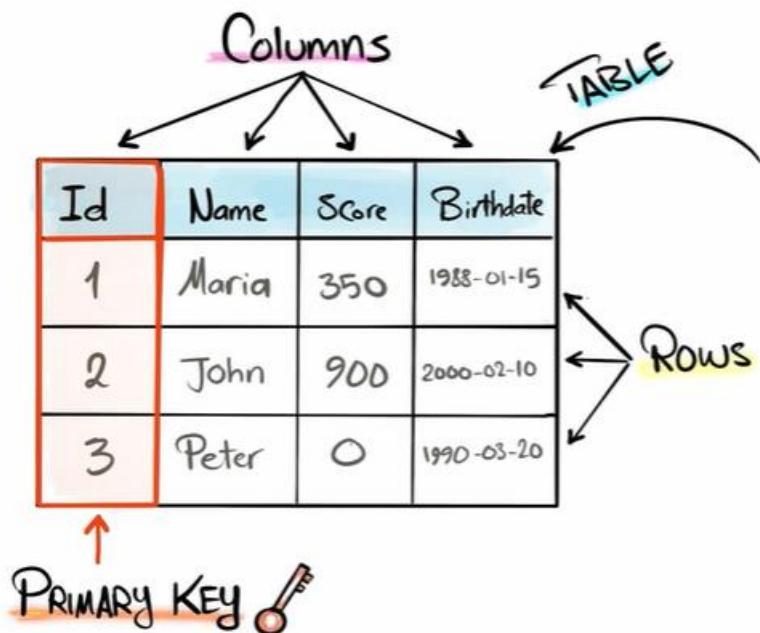
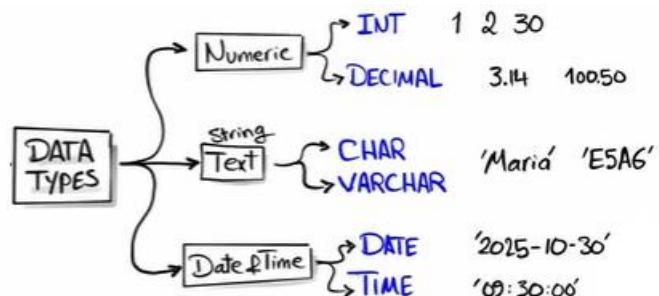


Table: A table is like a spreadsheet which organizes data into columns and rows.

Columns- are also called as fields. The column defines the data that you store.

Rows- are also called as records. The row is actually where data stores. In above example each row represents one customer info.

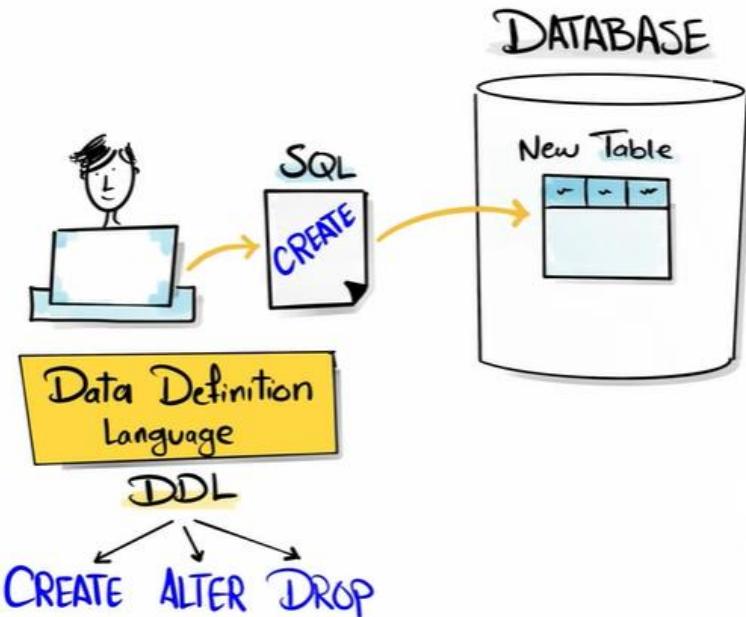
Primary key: A unique identifier for each row.



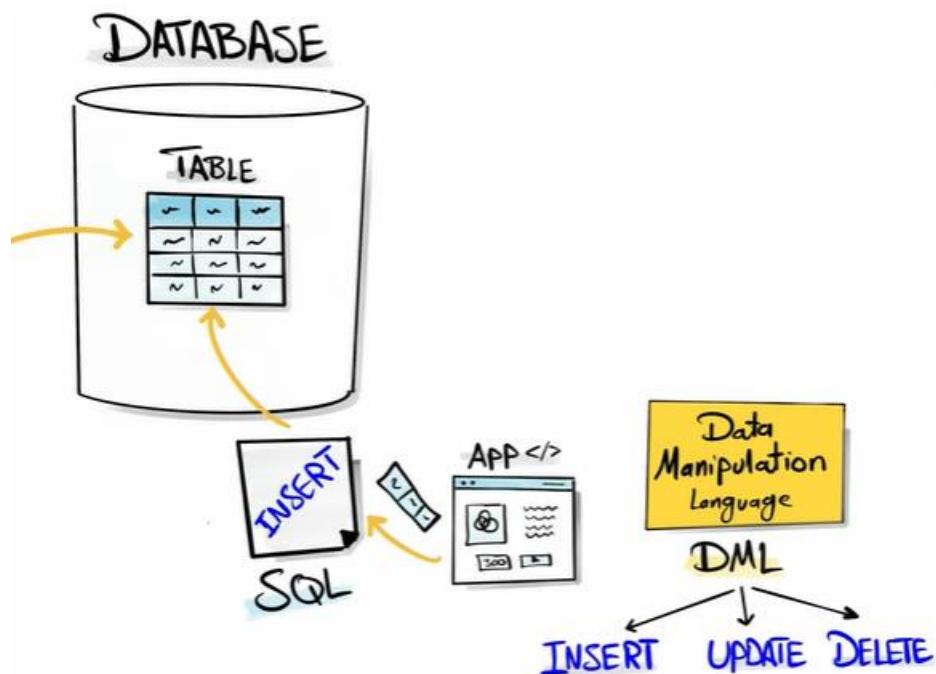
Data type: What kind of data we are storing.

Types of SQL Commands

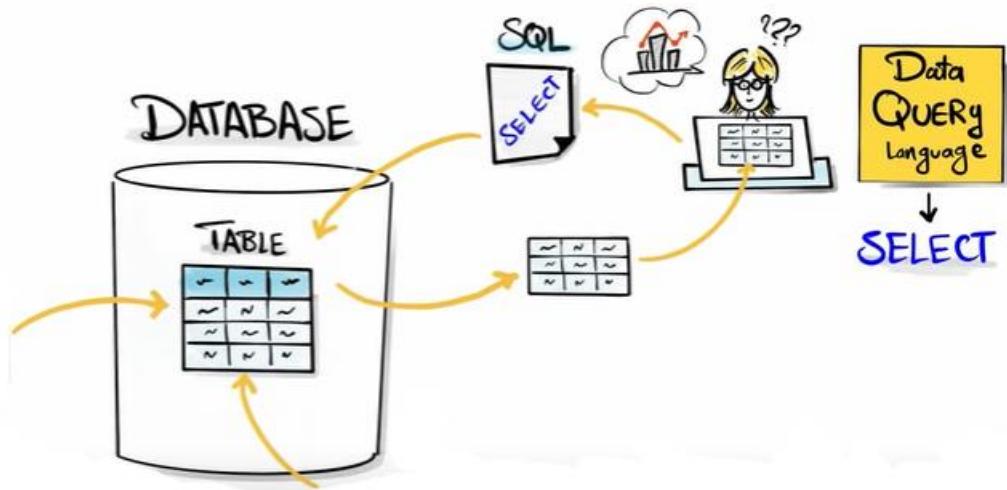
Data Definition language – used to perform actions related to table-create/modify/delete a table.



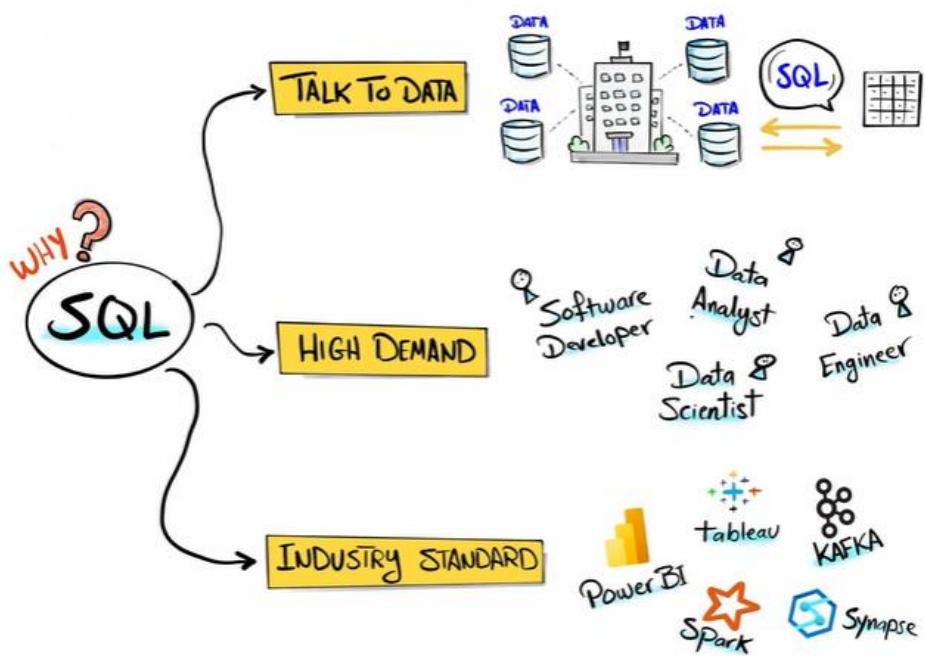
Data manipulation language – used to perform actions related to rows – Insert/Update/Delete a row.



Data query Language – query to get results.



Why SQL



Basic level

Create database

- 1) Right click Database>Click on New Database>Provide database name.
- 2) Using script to create database.

```
USE master;
```

```

GO

-- Drop and recreate the 'Database_name' database
IF EXISTS (SELECT 1 FROM sys.databases WHERE name = 'Database_name')
BEGIN
    ALTER DATABASE Database_name SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
    DROP DATABASE Database_name;
END;
GO

-- Create the 'Database_name' database
CREATE DATABASE Database_name;

```

Restore database

Url: <https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms>

Download .bak file.

Place the file in the location: C:\Program Files\Microsoft SQL Server\MSSQL16.SQLEXPRESS\MSSQL\Backup

Open SSMS >Right click on Database>Click on Restore Database>Select device>Select database>Click Ok.

SQL Components

SQL Statement

```

SQL STATEMENT
-- Retrieve Customers Data
SELECT
    name,
    LOWER(country)
FROM customers
WHERE country = 'Italy'

```

A SQL statement is a single command or query written in the SQL (Structured Query Language) syntax to interact with a database. SQL statements are used to perform various tasks, such as retrieving, inserting, updating, deleting, or managing data within a database.

SQL Comments

```
-- Retrieve Customers Data
SELECT
    name,
    LOWER(country)
FROM customers
WHERE country = 'Italy'
```

In SQL, comments are annotations or notes that you can add to your code to make it easier to understand. They are not executed by the database engine and are purely meant for human readability. Comments are particularly helpful when collaborating with others or when you revisit your code after some time.

There are two main types of comments in SQL:

- 1) **Single-Line Comments:** Start with -- (double hyphen). Anything after these symbols on the same line is considered a comment.
- 2) **Multi-Line Comments:** Enclosed between /* and */. These can span multiple lines.

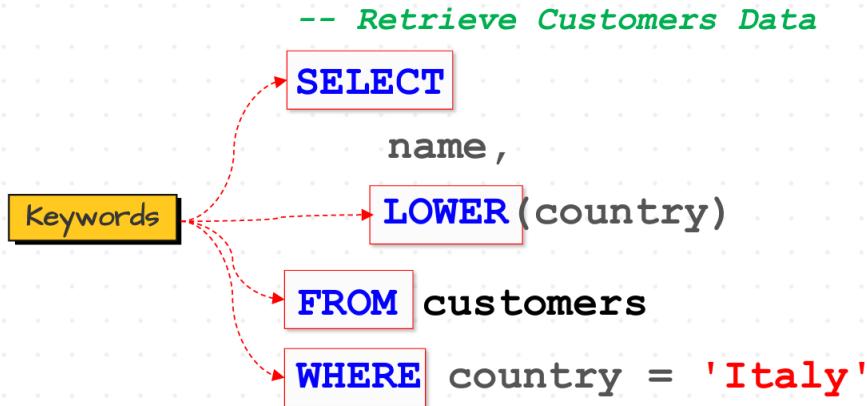
SQL Clause

```
-- Retrieve Customers Data
SELECT
    name,
    LOWER(country)
FROM customers
WHERE country = 'Italy'
```

A SQL clause is a component of a SQL statement used to define specific conditions or modify its behaviour. Clauses help refine queries, control data manipulation, and manage database interactions more effectively.

SELECT	DISTINCT	CASE
FROM	UNION / UNION ALL	SET
WHERE	EXCEPT	ALTER
ORDER BY	INTERSECT	INSERT INTO
GROUP BY	IN	DELETE
HAVING	BETWEEN	TRUNCATE
JOIN (INNER, LEFT, RIGHT, FULL OUTER)	LIKE	WITH (Common Table Expressions, or CTEs)
LIMIT / TOP / FETCH	IS NULL / IS NOT NULL	

Keywords



Keywords in SQL are reserved words that have special meanings and are part of the language syntax. They cannot be used as identifiers (like table names, column names, etc.) unless enclosed in quotes.

ADD	DELETE	INNER	PRIMARY
ALL	DESC	INSERT	REFERENCES
ALTER	DISTINCT	INTERSECT	REVOKE
AND	DROP	INTO	RIGHT
ANY	ELSE	IS	ROLLBACK
AS	EXISTS	JOIN	SELECT
ASC	FETCH	KEY	SET
BETWEEN	FOR	LEFT	TABLE
BY	FOREIGN	LIKE	TOP
CASE	FROM	LIMIT	TRUNCATE
CHECK	FULL	NOT	UNION
COLUMN	GRANT	NULL	UNIQUE
COMMIT	GROUP	ON	UPDATE
CONSTRAINT	HAVING	OR	VALUES
CREATE	IN	ORDER	VIEW
DEFAULT	INDEX	OUTER	WHERE
WITH			

Functions

-- Retrieve Customers Data

SELECT

Function

name ,

LOWER(country)

FROM customers

WHERE country = 'Italy'

Functions in SQL are built-in operations that allow you to manipulate and process data effectively. They can perform calculations, modify values, and analyze data. SQL functions are broadly categorized into:

1. Aggregate Functions

COUNT()
SUM()
AVG()
MAX()
MIN()

2. Scalar Functions

UPPER()
LOWER()
LENGTH() / LEN()
ROUND()

3. String Functions

SUBSTRING() / MID()
TRIM()
REPLACE()
CONCAT()
CHARINDEX() / INSTR()

4. Date and Time Functions

GETDATE() / NOW()
DATEPART()
YEAR(), MONTH(), DAY()
DATENAME()
DATEDIFF()
DATEADD()

5. Mathematical Functions

ABS()
POWER()
SQRT()
ROUND()
CEIL() / CEILING()
FLOOR()

6. Conversion Functions

Cast()
Convert()
Format()

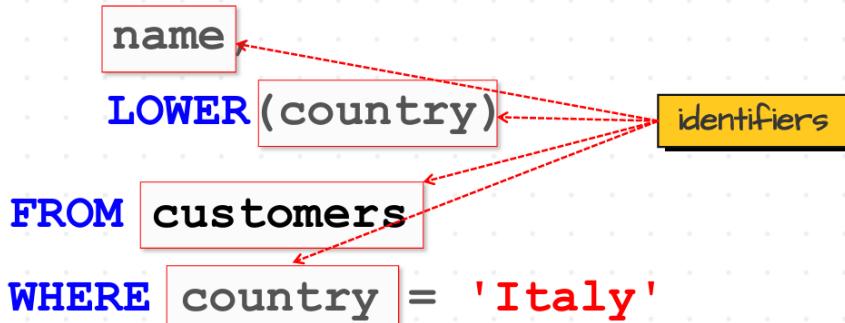
7. Others

COALESCE()
ISNULL() / IFNULL()
CASE()

Identifiers

-- Retrieve Customers Data

SELECT



Identifiers in SQL are the names used to represent database objects like tables, columns, indexes, views, procedures, and more. They serve as unique labels that help identify and refer to these objects in SQL queries and commands.

1. Rules for Naming:

- An identifier typically starts with a letter (A-Z or a-z) or an underscore (_), followed by letters, numbers (0-9), or underscores.
- Special characters (e.g., @, \$, #) and spaces are generally not allowed unless enclosed in double quotes ("") or square brackets ([])) in some systems.
- Reserved keywords in SQL (e.g., SELECT, WHERE) cannot be used as identifiers unless enclosed.

2. Case Sensitivity:

- SQL identifiers' case sensitivity depends on the database system. For instance, SQL Server treats identifiers as case-insensitive by default, whereas PostgreSQL treats them as case-sensitive when enclosed in double quotes.

3. Types of Identifiers:

- **Regular Identifiers:** Follow the standard naming conventions and do not require quotes.
- **Delimited Identifiers:** Enclosed in double quotes ("") or square brackets ([]), allowing the use of special characters and reserved keywords.

4. Naming Best Practices:

- Use meaningful names (e.g., user_id, order_date) to improve code readability.
- Stick to a consistent naming convention, like snake_case (order_details) or camelCase (orderDetails).

Operators

-- Retrieve Customers Data

SELECT

```
name ,  
LOWER(country)  
FROM customers  
WHERE country = 'Italy'
```

Operator

SQL operators are symbols or keywords used to perform various operations on data stored in a database. They help manipulate and query data efficiently. Here's an overview of the types of operators in SQL:

1. Arithmetic Operators

These are used for mathematical computations:

- + (Addition)
- (Subtraction)
- * (Multiplication)
- / (Division)
- % (Modulo)

2. Comparison Operators

These are used to compare two values:

- = (Equal to)
- <> or != (Not equal to)
- > (Greater than)
- < (Less than)
- >= (Greater than or equal to)
- <= (Less than or equal to)

3. Logical Operators

These combine multiple conditions:

- AND (Both conditions must be true)
- OR (At least one condition must be true)
- NOT (Negates a condition)

4. Bitwise Operators

Operate at the binary level:

- & (Bitwise AND)
- | (Bitwise OR)
- ^ (Bitwise XOR)

5. Set Operators

These are used to combine results of two queries:

6. Other Operators

LIKE: Used for pattern matching (e.g., WHERE name LIKE 'A%').

UNION (Combines results, removes duplicates)

UNION ALL (Combines results, keeps duplicates)

INTERSECT (Returns common results)

EXCEPT (Returns results from the first query not present in the second)

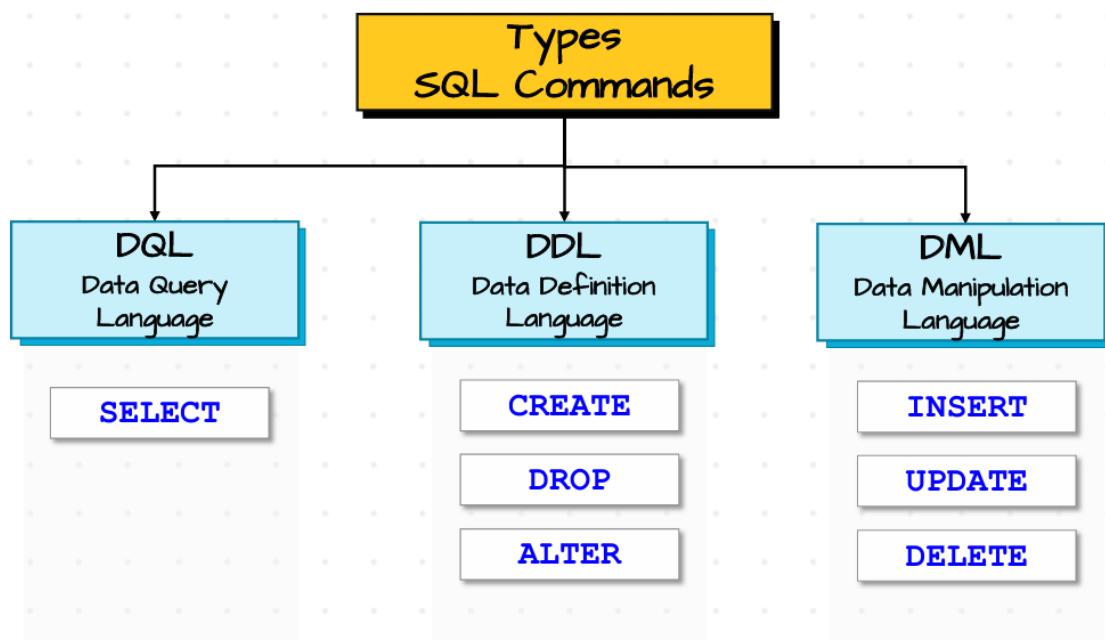
IN: Checks if a value exists in a list (e.g., WHERE id IN (1, 2, 3)).

BETWEEN: Checks if a value falls within a range (e.g., WHERE age BETWEEN 18 AND 25).

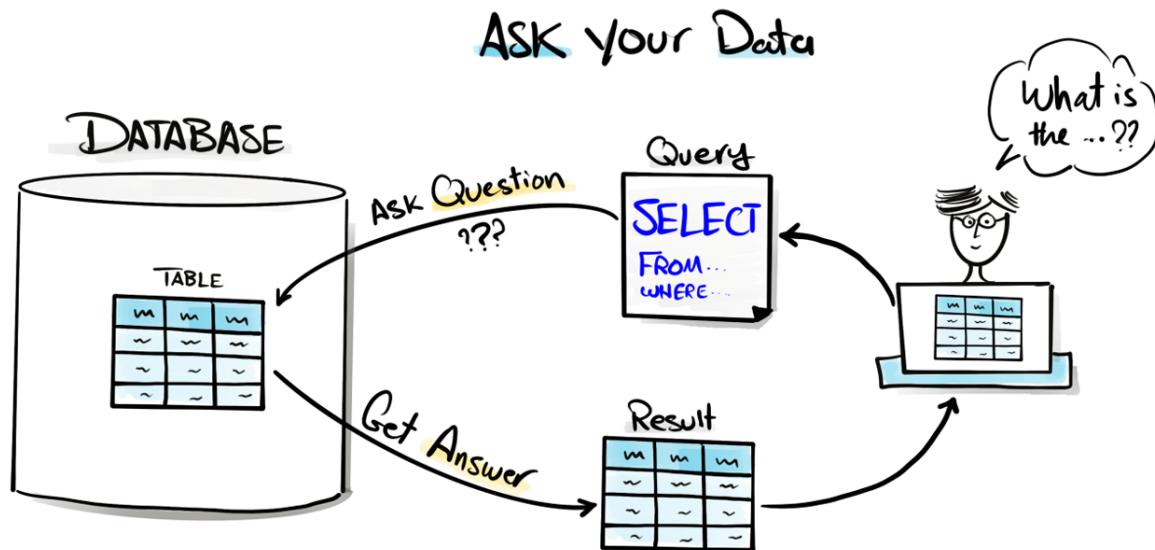
IS NULL: Tests for null values.

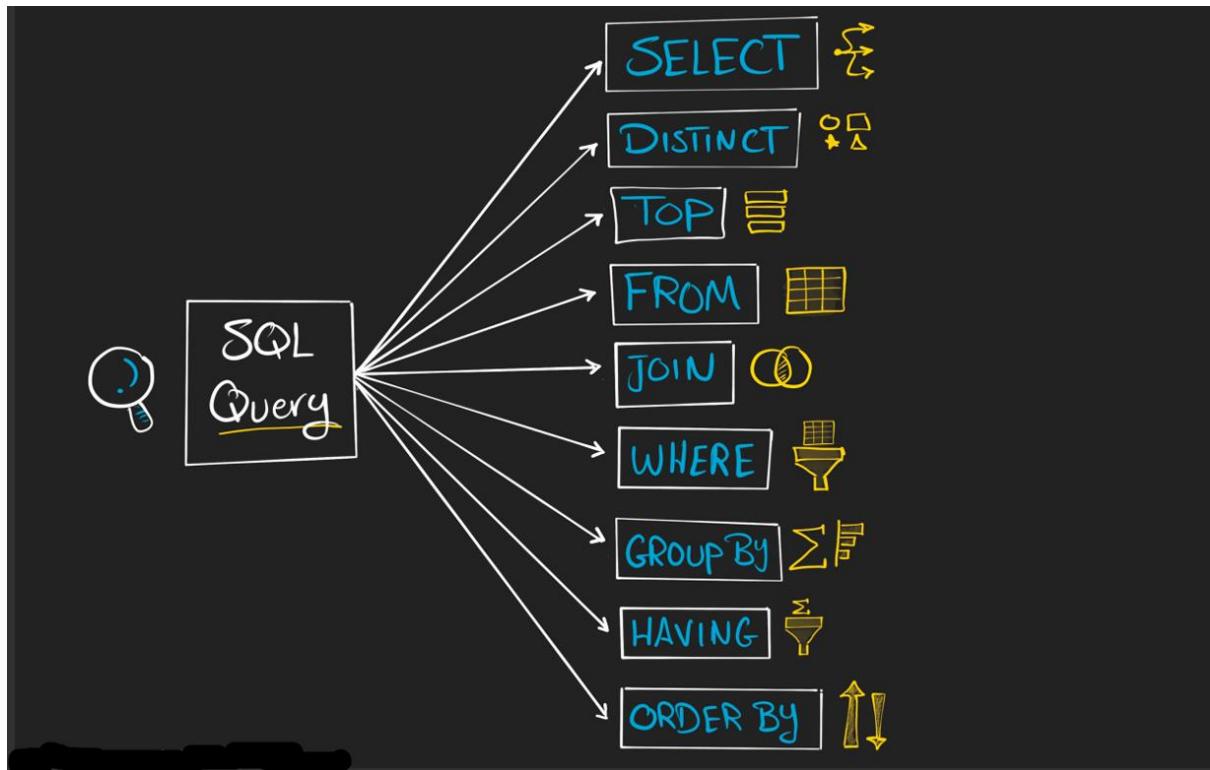
EXISTS: Checks if a subquery returns any rows.

Types of SQL Commands



Data Query Language





1) Select all the columns

Database 

	<u>id</u>	<u>name</u>	<u>Country</u>	<u>Score</u>
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

Select * (All)

Retrieves All Columns (Everything)

From

Tells SQL Where to find your Data

SELECT *

FROM Table

Database

	<u>id</u>	<u>name</u>	<u>Country</u>	<u>Score</u>
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

Keep All Columns!!

	<u>id</u>	<u>name</u>	<u>Country</u>	<u>Score</u>
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

① FROM Table
② SELECT *

2) Select only mentioned columns in the SQL Statement

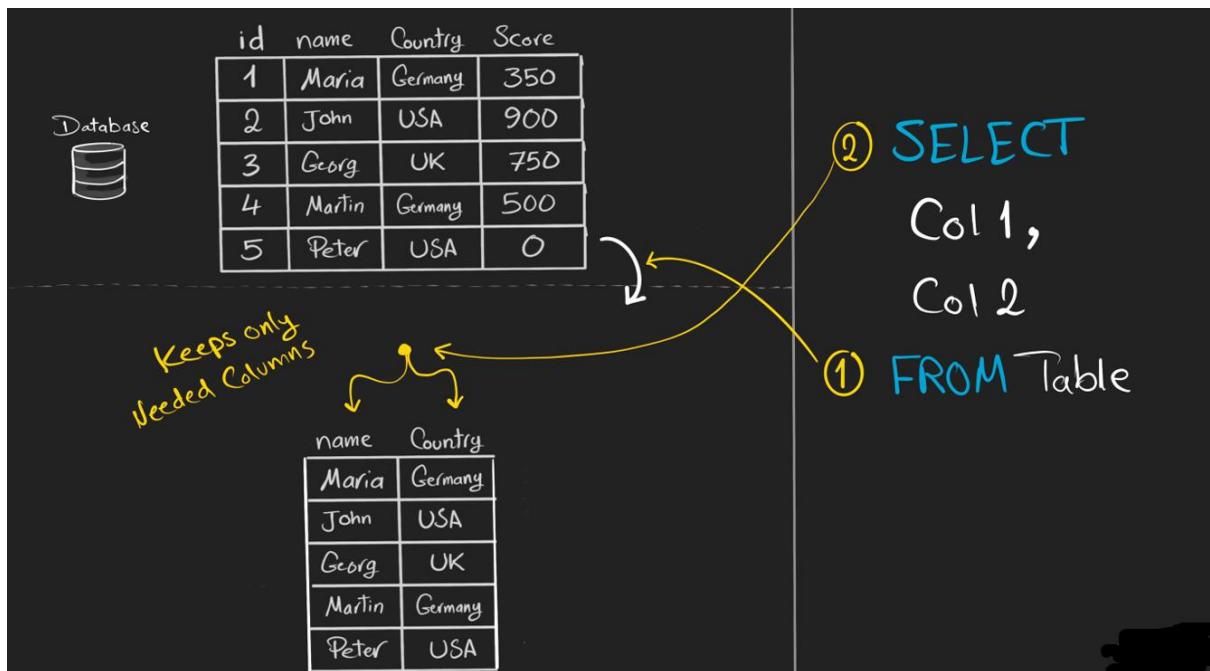
Database

	<u>id</u>	<u>name</u>	<u>Country</u>	<u>Score</u>
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

Select Few Columns

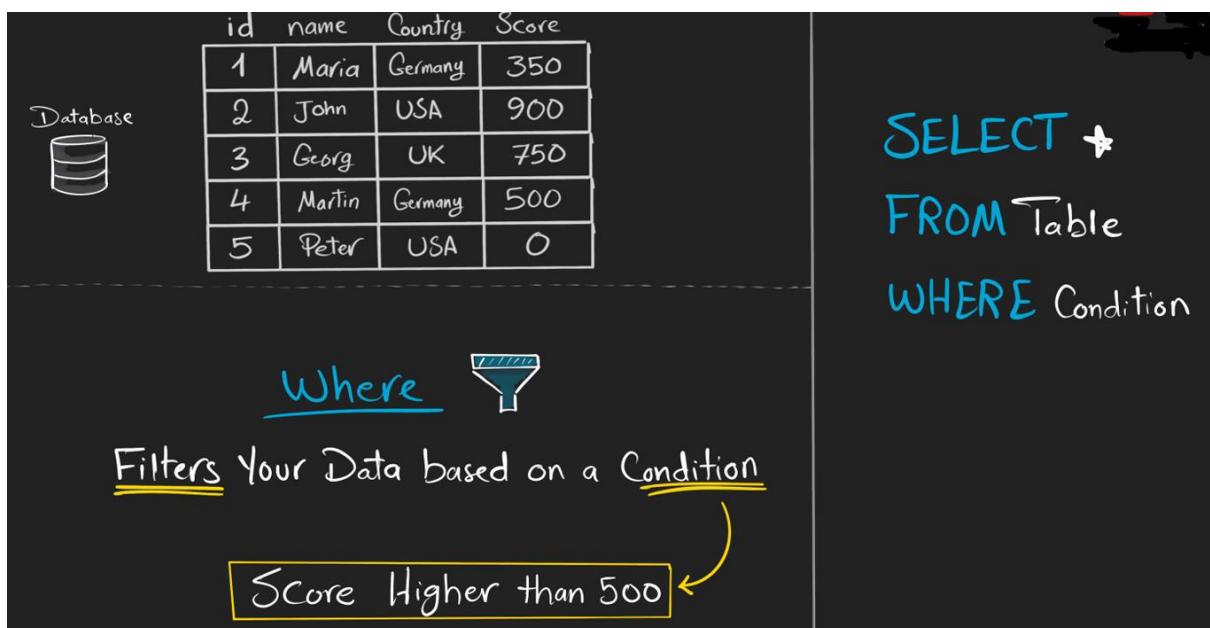
Pick only the Columns You Need
instead of All.

SELECT
Col 1,
Col 2
FROM Table



SQL Order of execution

Using where clause



Database

	<u>id</u>	<u>name</u>	<u>Country</u>	<u>Score</u>
	1	Maria	Germany	350
	2	John	USA	900
	3	Georg	UK	750
	4	Martin	Germany	500
	5	Peter	USA	0

	<u>id</u>	<u>name</u>	<u>Country</u>	<u>Score</u>
	1	Maria	Germany	350
	2	John	USA	900
	3	Georg	UK	750
	4	Martin	Germany	500
	5	Peter	USA	0

- ③ **SELECT ***
- ① **FROM Table**
- ② **WHERE Condition**

Using Order by clause

Database

	<u>id</u>	<u>name</u>	<u>Country</u>	<u>Score</u>
	1	Maria	Germany	350
	2	John	USA	900
	3	Georg	UK	750
	4	Martin	Germany	500
	5	Peter	USA	0

Order By

Sort Your Data

ASC ↗ Lowest ↑
 Highest ↓

DESC ↘ Highest ↑
 Lowest ↓

SELECT *
FROM Table
ORDER BY Score DESC

Database

	id	name	Country	Score
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

	id	name	Country	Score
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
1	Maria	Germany	350	
5	Peter	USA	0	

↓

Highest

Lowest

③ SELECT *

① FROM Table

② ORDER BY Score DESC

Order by 2 columns

Database

	id	name	Country	Score
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

	id	name	Country	Score
4	Martin	Germany	500	
1	Maria	Germany	350	
3	Georg	UK	750	
2	John	USA	900	
5	Peter	USA	0	

↓ ↓

Highest

Lowest

Highest

Lowest

Highest

Lowest

③ SELECT *

① FROM Table

② ORDER BY

Country ASC,
Score DESC

Using Group by clause

Database

	id	name	Country	Score
1	1	Maria	Germany	350
2	2	John	USA	900
3	3	Georg	UK	750
4	4	Martin	Germany	500
5	5	Peter	USA	0

Group By
Combines rows with the same value
Aggregates a Column By another Column
Total Score By Country

SELECT Category
Country, Aggregation
SUM(score)
FROM Table
GROUP BY Country

Database

	id	name	Country	Score
1	1	Maria	Germany	350
2	2	John	USA	900
3	3	Georg	UK	750
4	4	Martin	Germany	500
5	5	Peter	USA	0

③ SELECT
Country,
SUM(score)
① FROM Table
② GROUP By Country

↓ \sum

1	Germany	850
2	USA	900
3	UK	750

Using Having

Database

	id	name	Country	Score
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

Having
Filters Data After Aggregation
 Can be used only with Group By

SELECT
 Country,
 SUM(score)
FROM Table
GROUP BY Country
HAVING $\text{SUM(score)} > 800$

Database

	id	name	Country	Score
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

\sum

Germany	850
USA	900

Total Score > 800

④ SELECT
 Country,
 SUM(score)
① FROM Table
② GROUP By Country
③ HAVING $\text{SUM(score)} > 800$

Database

	id	name	Country	Score
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

Filter Your Data

① Before Aggregation → **WHERE Score > 400**
② After Aggregation → **HAVING $\text{SUM(score)} > 800$**

SELECT
 Country,
 SUM(score)
FROM Table
WHERE Score > 400
GROUP By Country
HAVING $\text{SUM(score)} > 800$

To select Distinct values

Database

	id	name	Country	Score
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

Distinct

Removes Duplicates (Repeated Values)

each Value appears only Once

SELECT DISTINCT
Col
FROM Table

Database

	id	name	Country	Score
1	Maria	Germany	350	
2	John	USA	900	
3	Georg	UK	750	
4	Martin	Germany	500	
5	Peter	USA	0	

Country

Country
Germany
USA
UK
Germany
USA

③
② SELECT DISTINCT
Country
① FROM Table

The diagram illustrates the process of selecting distinct values. It starts with a database table containing five rows of data. A specific column, 'Country', is selected and extracted into a separate list. This list contains five entries: Germany, USA, UK, Germany, and USA. A funnel-shaped filter is applied to this list, with the condition 'only Once!', which removes the duplicate entries. The resulting list contains three unique entries: Germany, USA, and UK. This final list is then used in a SELECT DISTINCT query, as indicated by the numbered steps on the right.

To select Top n rows

A diagram illustrating the concept of selecting top n rows from a database. On the left, a database icon is labeled "Database". Next to it is a table with columns "id", "name", "Country", and "Score", containing 5 rows of data. Below the table is the handwritten text "TOP (Limit)". To the right of the table is a funnel icon. Further right, the SQL query "SELECT TOP 3" is written, followed by an asterisk and "FROM Table". At the bottom, the text "Restrict the Number of Rows Returned" is written.

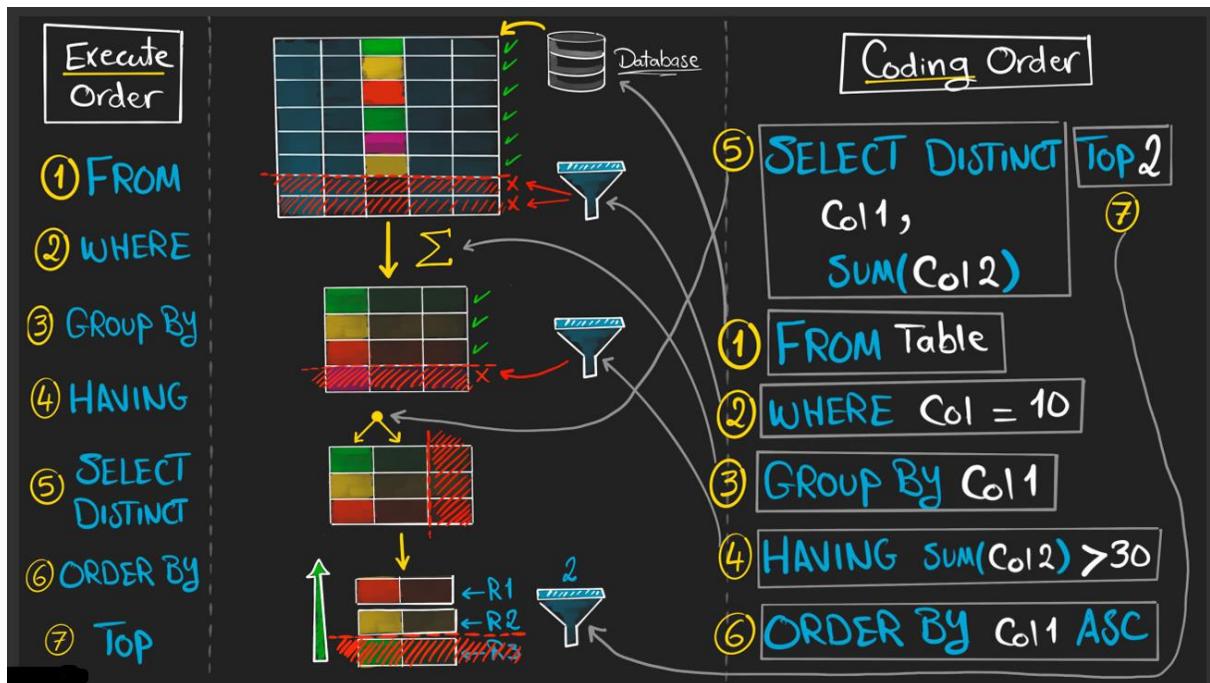
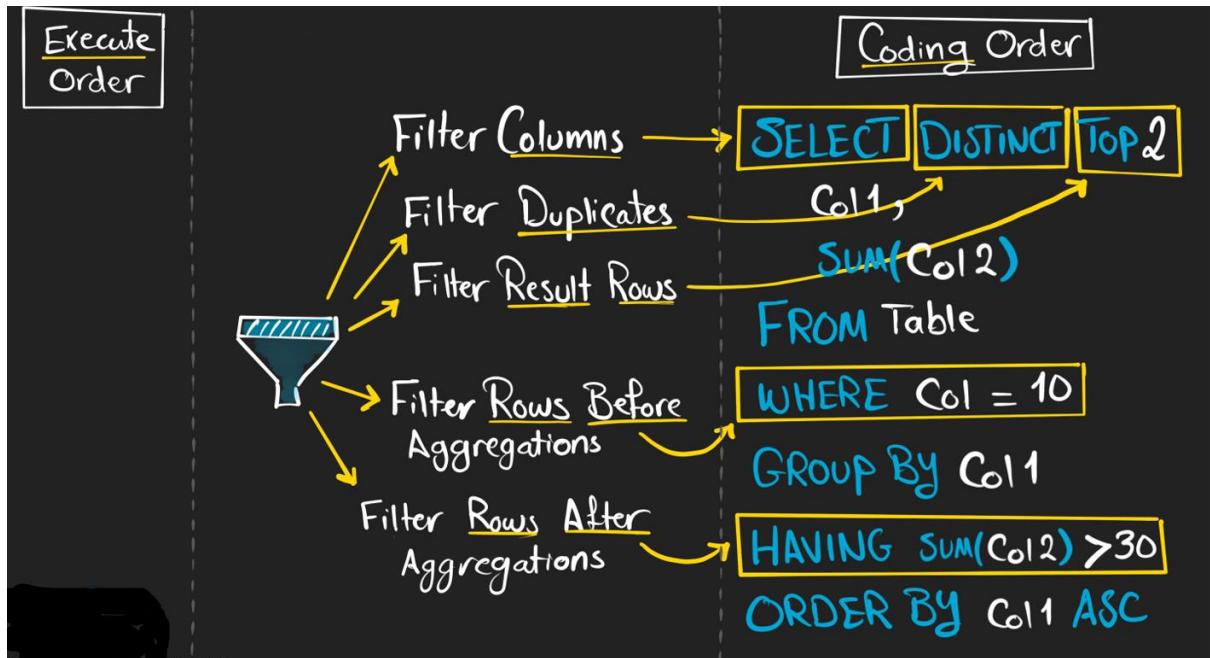
id	name	Country	Score
1	Maria	Germany	350
2	John	USA	900
3	Georg	UK	750
4	Martin	Germany	500
5	Peter	USA	0

A diagram illustrating the execution flow of a SELECT TOP 3 query. On the left, a database icon is labeled "Database". Above it is a table with 5 rows. A yellow arrow points from the table down to a smaller table below, which contains only the first three rows. This smaller table is labeled "Row 1 →", "Row 2 →", and "Row 3 →". To the right of the tables, the SQL query "② SELECT TOP 3" is written above an asterisk and "① FROM Table". A large yellow circle encloses the entire process, with a number "3" at the top right of the circle.

id	name	Country	Score
1	Maria	Germany	350
2	John	USA	900
3	Georg	UK	750
4	Martin	Germany	500
5	Peter	USA	0

id	name	Country	Score
1	Maria	Germany	350
2	John	USA	900
3	Georg	UK	750

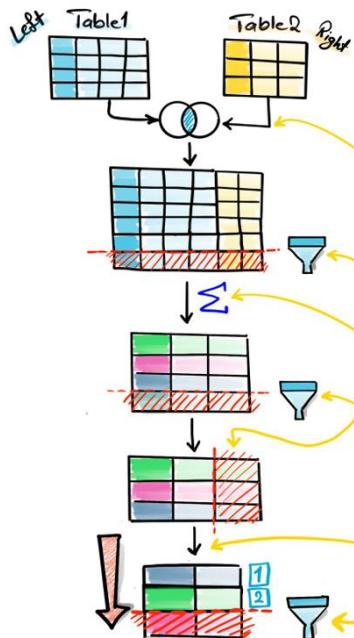
Execution order



Complete execution order

Execution Order

- ① FROM JOIN
- ② WHERE
- ③ GROUP BY
- ④ HAVING
- ⑤ SELECT DISTINCT
- ⑥ ORDER BY
- ⑦ TOP



Coding Order

`SELECT DISTINCT
a.col,
SUM(b.col)`

`FROM Table1 AS a
JOIN Table2 AS b
ON a.id = b.id`

`WHERE a.col = 10`

`GROUP BY a.col`

`HAVING SUM(b.col)`

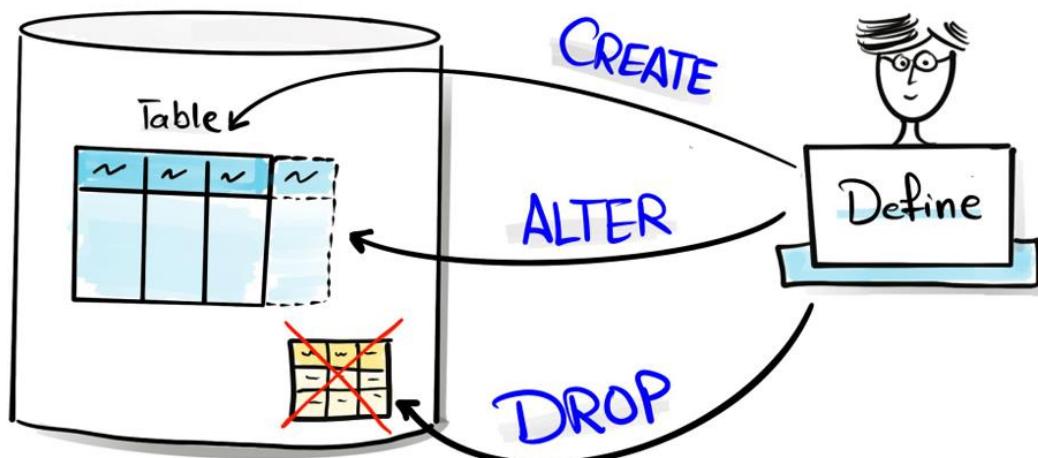
`ORDER BY a.col`

`Top 2`

Data Definition Language

Define Structure of Your Data

DATABASE



Create a table

```
[-/* Create a new table called persons  
with columns: id, person_name, birth_date, and phone */  
  
[-CREATE TABLE persons (  
    id INT NOT NULL,  
    person_name VARCHAR(50) NOT NULL,  
    birth_date DATE,  
    phone VARCHAR(15) NOT NULL,  
    CONSTRAINT pkpersons PRIMARY KEY (id)  
)
```

Alter a table

1

```
-- Add a new column called email to the persons table
```

```
[-ALTER TABLE persons  
[- ADD email VARCHAR(50) NOT NULL
```

2

```
-- Remove the column phone from the persons table
```

```
[-ALTER TABLE persons  
[- DROP COLUMN phone
```

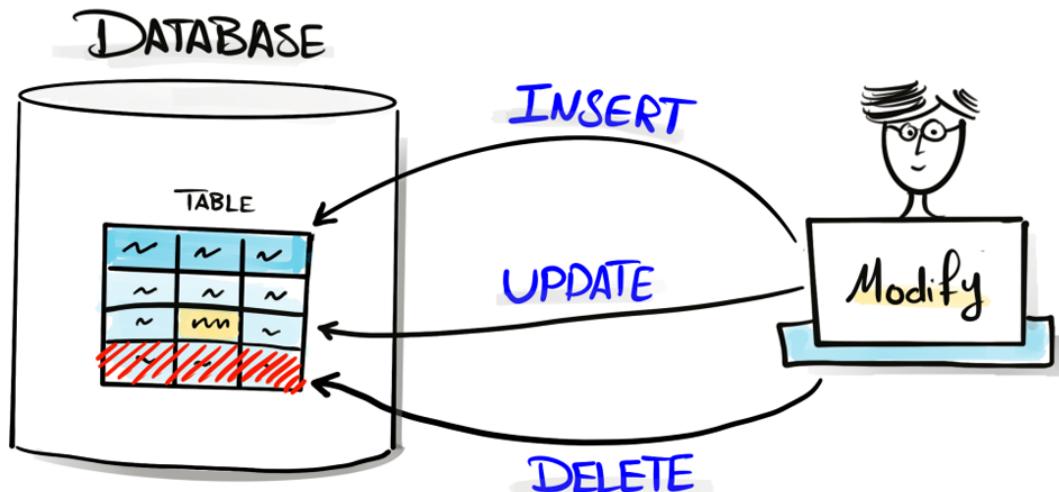
Drop a table

```
-- Delete the table persons from the database
```

```
DROP TABLE persons
```

Data Manipulation Language

Modify (Manipulate) Your Data



INSERT Syntax

OPTIONAL: If no columns are specified, SQL expects values for all columns

```
INSERT INTO table_name (column1, column2, column3,...)  
VALUES (value1, value2, value3,...)  
, (value1, value2, value3,...) ----- Multiple inserts
```

Rule

- Match the number of columns and Values.

```
INSERT INTO employees (id, name, position, salary)
```

```
VALUES
```

```
(1, 'Alice Johnson', 'Manager', 75000),  
(2, 'Bob Smith', 'Developer', 60000),  
(3, 'Charlie Brown', 'Analyst', 50000),  
(4, 'Diana Green', 'Designer', 55000);
```

UPDATE Syntax

```
UPDATE table_name
      SET column1 = value1,
          column2 = value2
      WHERE <condition>
```

NOTE

- Always use WHERE to avoid UPDATING all rows unintentionally

UPDATE employees

SET salary = 70000

WHERE name = 'Charlie Brown';

DELETE Syntax

```
DELETE FROM table_name
      WHERE <condition>
```

NOTE

- Always use WHERE to avoid DELETING all rows unintentionally

DELETE FROM employees

WHERE id = 3;