| **Basics** | **File System** | **DBMS** |
| --- | --- | --- |
| **Structure** | The file system is a way of arranging the files in a storage medium within a computer. | DBMS is software for managing the database. |
| **Data Redundancy** | Redundant data can be present in a file system. | In DBMS there is no redundant data. |
| **Backup and Recovery** | It doesn't provide Inbuilt mechanism for backup and recovery of data if it is lost. | It provides in house tools for backup and recovery of data even if it is lost. |
| **Query processing** | There is no efficient query processing in the file system. | Efficient query processing is there in DBMS. |
| **Consistency** | There is less data consistency in the file system. | There is more data consistency because of the process of [normalization](https://www.geeksforgeeks.org/dbms/normal-forms-in-dbms/). |
| **Complexity** | It is less complex as compared to DBMS. | It has more complexity in handling as compared to the file system. |
| **Security Constraints** | File systems provide less security in comparison to DBMS. | DBMS has more security mechanisms as compared to file systems. |
| **Cost** | It is less expensive than DBMS. | It has a comparatively higher cost than a file system. |
| **Data Independence** | There is no data independence. | In DBMS [data independence](https://www.geeksforgeeks.org/dbms/what-is-data-independence-in-dbms/)exists, mainly of two types:  1) [Logical Data Independence](https://www.geeksforgeeks.org/dbms/physical-and-logical-data-independence/).  2)Physical Data Independence. |
| **User Access** | Only one user can access data at a time. | Multiple users can access data at a time. |
| **Meaning** | The users are not required to write procedures. | The user has to write procedures for managing databases |
| **Sharing** | Data is distributed in many files. So, it is not easy to share data. | Due to centralized nature data sharing is easy |
| **Data Abstraction** | It give details of storage and representation of data | It hides the internal details of [Database](https://www.geeksforgeeks.org/dbms/what-is-database/) |
| **Integrity Constraints** | Integrity Constraints are difficult to implement | Integrity constraints are easy to implement |
| **Attribute *s*** | To access data in a file , user requires attributes such as file name, file location. | No such attributes are required. |
| **Example** | [Cobol](https://www.geeksforgeeks.org/installation-guide/how-to-install-cobol-on-macos/), [C++](https://www.geeksforgeeks.org/cpp/c-plus-plus/) | [Oracle](https://www.geeksforgeeks.org/interview-experiences/oracle-interview-experience-8/), [SQL Server](https://www.geeksforgeeks.org/sql/sql-tutorial/) |

Great question, Ram! Here's a clear comparison between **DBMS (Database Management System)** and **File Storage**:

**🔍 1. Definition**

* **File Storage**: A system where data is stored in files on a disk (e.g., .txt, .csv, .doc). It's simple and used for basic data storage.
* **DBMS**: A software system that uses structured formats (like tables) to store, manage, and retrieve data efficiently.

**📊 2. Data Organization**

* **File Storage**: Data is stored in a flat format, often requiring manual parsing.
* **DBMS**: Data is organized in tables with rows and columns, supporting relationships between data.

**🔄 3. Data Access & Retrieval**

* **File Storage**: Slower and less efficient; requires custom code to search or update data.
* **DBMS**: Fast and efficient; uses SQL (Structured Query Language) for querying and updating.

**🔐 4. Security**

* **File Storage**: Basic security (e.g., file permissions).
* **DBMS**: Advanced security features like user authentication, access control, and encryption.

**🔄 5. Data Integrity & Consistency**

* **File Storage**: Hard to maintain consistency across multiple files.
* **DBMS**: Ensures data integrity through constraints, transactions, and ACID properties.

**📈 6. Scalability**

* **File Storage**: Not ideal for large-scale applications.
* **DBMS**: Designed to handle large volumes of data and concurrent users.

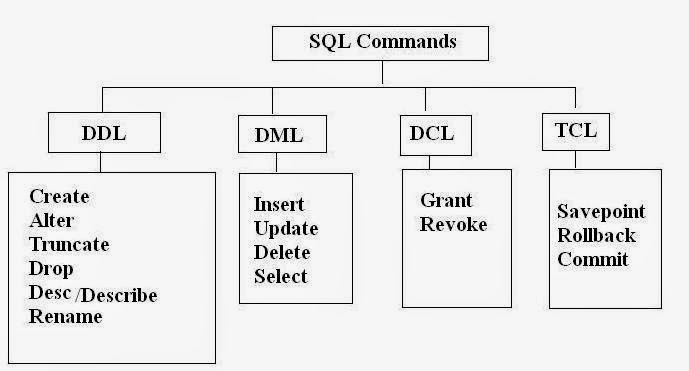
**🔧 7. Backup & Recovery**

* **File Storage**: Manual backup and recovery.
* **DBMS**: Built-in tools for automated backup and recovery.

**🧠 Use Case Examples**

* **File Storage**: Simple logs, configuration files, small-scale apps.
* **DBMS**: Banking systems, e-commerce platforms, enterprise applications.

Would you like a diagram or table to visualize this comparison? Or are you exploring this for a specific project or exam?



[This Photo](https://soran-abdulrahman.blogspot.com/2015/01/SQL-Tutorial-part1.html) by Unknown Author is licensed under [CC BY](https://creativecommons.org/licenses/by/3.0/)

DDL Comands

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| [CREATE](https://www.geeksforgeeks.org/sql/sql-create-table/) | Create database or its objects (table, index, function, views, store procedure and triggers) | CREATE TABLE table\_name (column1 data\_type, column2 data\_type, ...); |
| [DROP](https://www.geeksforgeeks.org/sql/sql-drop-truncate/) | Delete objects from the database | DROP TABLE table\_name; |
| [ALTER](https://www.geeksforgeeks.org/sql/sql-alter-add-drop-modify/) | Alter the structure of the database | ALTER TABLE table\_name ADD COLUMN column\_name data\_type; |
| [TRUNCATE](https://www.geeksforgeeks.org/sql/sql-drop-truncate/) | Remove all records from a table, including all spaces allocated for the records are removed | TRUNCATE TABLE table\_name; |
| [COMMENT](https://www.geeksforgeeks.org/sql/sql-comments/) | Add comments to the data dictionary | COMMENT ON TABLE table\_name IS 'comment\_text'; |
| [RENAME](https://www.geeksforgeeks.org/sql/sql-rename-table/) | Rename an object existing in the database | RENAME TABLE old\_table\_name TO new\_table\_name; |

**Notes:**

* **Truncate** will delete only attributes whereas **drop** will delete whole with its structure.
* **Inner join, Outer Join, Left Join, Right Join**

**📘 SQL Learning Roadmap**

**🟢 1. Introduction to SQL & Databases**

* What is SQL?
* What is a Relational Database?
* SQL vs NoSQL (brief overview)
* Popular SQL databases (MySQL, PostgreSQL, SQLite, SQL Server, Oracle)

**🟢 2. Basic SQL Syntax**

* SQL statements: SELECT, FROM
* Case sensitivity
* Comments in SQL
* Query structure

**🟢 3. Data Retrieval (SELECT Queries)**

* SELECT specific columns
* SELECT \* (all columns)
* Aliases with AS
* Limiting results with LIMIT
* Ordering results with ORDER BY
* Removing duplicates with DISTINCT

**🟢 4. Filtering Data**

* WHERE clause
* Comparison operators (=, !=, >, <, etc.)
* Logical operators (AND, OR, NOT)
* BETWEEN, IN, LIKE, IS NULL

**🟢 5. Working with Multiple Tables**

* Introduction to **Joins**
  + INNER JOIN
  + LEFT JOIN / RIGHT JOIN
  + FULL OUTER JOIN
* Cross Join
* Self Join

**🟢 6. Aggregation & Grouping**

* Aggregate functions: COUNT(), SUM(), AVG(), MIN(), MAX()
* GROUP BY
* HAVING vs WHERE

**🟢 7. Subqueries & Nested Queries**

* Scalar subqueries
* IN and EXISTS with subqueries
* Correlated subqueries

**🟢 8. Data Modification**

* INSERT INTO
* UPDATE
* DELETE
* TRUNCATE

**🟢 9. Table Management**

* CREATE TABLE
* Data types
* ALTER TABLE
* DROP TABLE

**🟢 10. Constraints and Keys**

* Primary Key
* Foreign Key
* Unique
* Not Null
* Default
* Check

**🟢 11. Indexes and Performance**

* What is an Index?
* Creating and dropping indexes
* When to use indexes

**🟢 12. Views**

* Creating and using VIEW
* Updating through views
* Dropping views

**🟢 13. SQL Functions**

* String functions (UPPER(), LOWER(), CONCAT(), etc.)
* Numeric functions (ROUND(), ABS(), etc.)
* Date functions (NOW(), DATE\_ADD(), etc.)
* Conversion functions (CAST(), CONVERT())

**🟢 14. Transactions and ACID Properties**

* BEGIN, COMMIT, ROLLBACK
* Understanding ACID (Atomicity, Consistency, Isolation, Durability)

**🟢 15. Advanced SQL (Optional)**

* Window functions (ROW\_NUMBER(), RANK(), LEAD(), etc.)
* Common Table Expressions (CTEs)
* Recursive queries
* Pivot and Unpivot

Perfect! We’ll go topic by topic, with simple explanations, examples, and code you can run in most SQL platforms (e.g., MySQL, PostgreSQL, SQLite). You can also use free online tools like:

* SQLFiddle
* DB-Fiddle
* Mode SQL Tutorial
* W3Schools SQL

**🟢 Topic 1: Introduction to SQL & Databases**

**✅ What is SQL?**

**SQL (Structured Query Language)** is a language used to communicate with **relational databases**. It allows you to **create**, **read**, **update**, and **delete** data—this is commonly known as **CRUD**.

**✅ What is a Relational Database?**

A **Relational Database** stores data in **tables** (like Excel spreadsheets). Each table has **rows** and **columns**.

Example:

| **ID** | **Name** | **Age** |
| --- | --- | --- |
| 1 | Alice | 25 |
| 2 | Bob | 30 |

This is a table named People.

**✅ What can SQL do?**

* Query data from databases
* Insert, update, and delete records
* Create new tables and databases
* Set permissions and constraints
* Combine data from multiple tables (JOINs)

**✅ Popular SQL Databases**

* **MySQL** – Very popular and open-source
* **PostgreSQL** – Advanced features, open-source
* **SQLite** – Lightweight, file-based
* **SQL Server** – Microsoft’s RDBMS
* **Oracle DB** – Enterprise-level features

**✅ Basic SQL Syntax**

Let’s look at a simple SQL query:

SELECT Name, Age FROM People WHERE Age > 20;

🔍 **Explanation**:

* SELECT → tells SQL what columns you want
* FROM → which table to get data from
* WHERE → adds a condition (only people older than 20)

**✅ Example: Creating a Table**

CREATE TABLE People ( ID INT, Name VARCHAR(100), Age INT );

✅ This creates a table named People with 3 columns: ID, Name, and Age.

**✅ Example: Inserting Data**

INSERT INTO People (ID, Name, Age) VALUES (1, 'Alice', 25), (2, 'Bob', 30);

✅ This adds two rows to the People table.

**✅ Example: Retrieving Data**

SELECT \* FROM People;

✅ This gets all data from the People table.

**✅ Summary**

| **Concept** | **SQL Command** |
| --- | --- |
| Create table | CREATE TABLE |
| Add data | INSERT INTO |
| Get data | SELECT |
| Filter data | WHERE |
| Show all data | SELECT \* |

**✅ Quiz – Topic 1**

1. **What does the**SELECT**statement do in SQL?**

A) Deletes data  
B) Retrieves data  
C) Updates table structure  
D) Creates a new table

1. **Which of the following best describes a relational database?**

A) A collection of JSON files  
B) A flat text file with comma-separated values  
C) A collection of tables with rows and columns  
D) A graphical dashboard tool

1. **Which SQL command would you use to add new records into a table?**

A) INSERT INTO  
B) UPDATE  
C) SELECT  
D) DELETE

Reply with your answers (e.g., **1B, 2C, 3A**) and I’ll tell you how you did. Once you're ready, we’ll jump into **Topic 2: Basic SQL Syntax**.

**🟢 Topic 2: Basic SQL Syntax**

**✅ 1. Basic SELECT Statement**

The SELECT statement is used to **retrieve data** from a table.

SELECT column1, column2 FROM table\_name;

If you want **all columns**, use \*:

SELECT \* FROM People;

**✅ 2. Case Sensitivity**

* **SQL keywords** like SELECT, FROM, WHERE are **not case-sensitive**.
* But **table names and column names may be case-sensitive**, depending on the database.

Example (all these are valid):

SELECT name FROM people; select NAME from PEOPLE; Select Name From People;

✔ Best practice: write SQL keywords in **UPPERCASE** and table/column names in **lowercase or CamelCase**.

**✅ 3. Using Aliases (**AS**)**

You can rename columns or tables temporarily using AS.

SELECT Name AS PersonName, Age AS YearsOld FROM People;

This shows:

| **PersonName** | **YearsOld** |
| --- | --- |
| Alice | 25 |
| Bob | 30 |

**✅ 4. Adding Comments**

* **Single-line comment**: use --
* **Multi-line comment**: use /\* ... \*/

-- This is a comment SELECT \* FROM People; /\* This is a multi-line comment \*/ SELECT Name FROM People;

**✅ 5. Limiting Results**

Use LIMIT to return only a certain number of rows (works in MySQL, PostgreSQL, SQLite):

SELECT \* FROM People LIMIT 1;

In SQL Server, use TOP:

SELECT TOP 1 \* FROM People;

**✅ 6. Ordering Results with ORDER BY**

SELECT \* FROM People ORDER BY Age ASC; -- ASC for ascending (default), DESC for descending

Descending order:

SELECT \* FROM People ORDER BY Age DESC;

**🧪 Practice Table**

Here’s a table called People we'll use for examples:

| **ID** | **Name** | **Age** |
| --- | --- | --- |
| 1 | Alice | 25 |
| 2 | Bob | 30 |
| 3 | Carol | 22 |

**✅ Example Queries**

-- Get names and ages SELECT Name, Age FROM People; -- Get all data SELECT \* FROM People; -- Rename columns using AS SELECT Name AS FullName, Age AS Years FROM People; -- Order by Age SELECT \* FROM People ORDER BY Age DESC; -- Limit to top 2 SELECT \* FROM People LIMIT 2;

**✅ Summary**

| **Concept** | **SQL Syntax Example** |
| --- | --- |
| Select columns | SELECT Name, Age FROM People; |
| Select all | SELECT \* FROM People; |
| Rename column | SELECT Name AS FullName |
| Comment line | -- This is a comment |
| Limit rows | SELECT \* FROM People LIMIT 5; |
| Sort rows | ORDER BY Age DESC |

**✅ Question 1:**

Which SQL statement retrieves all columns from a table named Employees?

A) SELECT ALL FROM Employees;  
B) GET \* FROM Employees;  
C) SELECT \* FROM Employees;  
D) SHOW ALL Employees;

**✅ Question 2:**

What does the AS keyword do in a SELECT statement?

A) Sorts data  
B) Filters rows  
C) Renames a column or table  
D) Deletes duplicate rows

**✅ Question 3:**

What is the correct way to get the top 3 oldest people from a table People?

A) SELECT \* FROM People ORDER BY Age ASC LIMIT 3;  
B) SELECT \* FROM People ORDER BY Age DESC LIMIT 3;  
C) SELECT \* FROM People LIMIT 3 ORDER BY Age DESC;  
D) SELECT TOP 3 FROM People ORDER BY Age DESC;

**✅ Your Answers Review**

1. **Question 1:**SELECT \* FROM Employees;  
   ✅ **Your answer: C – Correct!**  
   This retrieves all columns from the Employees table.
2. **Question 2:**AS**is used to rename columns or tables**  
   ✅ **Your answer: C – Correct!**  
   For example: SELECT Name AS FullName FROM People;
3. **Question 3: Get top 3 oldest people**  
   ✅ **Correct answer: B —**SELECT \* FROM People ORDER BY Age DESC LIMIT 3;  
   🎯 **Your answer: B – Correct!**  
   ORDER BY Age DESC sorts from oldest to youngest, and LIMIT 3 picks the top 3.

✅✅✅ **Perfect score!** You nailed it.

**🟢 Topic 3: Data Retrieval (**SELECT**Queries)**

**✅ 1. Selecting Specific Columns**

You can choose which columns to retrieve using SELECT column\_name.

SELECT Name, Age FROM People;

Returns only the Name and Age columns.

**✅ 2. Selecting All Columns**

Use \* to select all columns:

SELECT \* FROM People;

**✅ 3. Using Aliases (**AS**)**

Rename columns to make them clearer in the result.

SELECT Name AS FullName, Age AS Years FROM People;

**✅ 4. Removing Duplicates with**DISTINCT

If your data has duplicate values and you want only unique results:

SELECT DISTINCT Age FROM People;

🎯 Example:

| **Age** |
| --- |
| 25 |
| 30 |
| 22 |
| 25 |

Using DISTINCT will return:

| **Age** |
| --- |
| 25 |
| 30 |
| 22 |

**✅ 5. Limiting Results with**LIMIT

To show only the top N results:

SELECT \* FROM People LIMIT 2;

In SQL Server, use:

SELECT TOP 2 \* FROM People;

**✅ 6. Ordering Results with**ORDER BY

Sort results in ascending (ASC) or descending (DESC) order.

SELECT \* FROM People ORDER BY Age DESC;

**🧪 Example Table:**People

| **ID** | **Name** | **Age** |
| --- | --- | --- |
| 1 | Alice | 25 |
| 2 | Bob | 30 |
| 3 | Carol | 22 |
| 4 | Alice | 25 |

**✅ Example Queries**

-- Get all names SELECT Name FROM People; -- Get unique names SELECT DISTINCT Name FROM People; -- Get names and sort by age (oldest first) SELECT Name, Age FROM People ORDER BY Age DESC; -- Get only first 2 people SELECT \* FROM People LIMIT 2; -- Rename Age column SELECT Name, Age AS YearsOld FROM People;

**✅ Summary**

| **Task** | **SQL Syntax Example** |
| --- | --- |
| Select all columns | SELECT \* FROM People; |
| Select specific columns | SELECT Name, Age FROM People; |
| Rename column | SELECT Age AS YearsOld FROM People; |
| Remove duplicates | SELECT DISTINCT Name FROM People; |
| Limit result rows | SELECT \* FROM People LIMIT 3; |
| Sort results | ORDER BY Age DESC |

**✅ Question 1:**

Which SQL query returns only the unique names from the People table?

A) SELECT ALL Name FROM People;  
B) SELECT DISTINCT Name FROM People;  
C) SELECT UNIQUE Name FROM People;  
D) SELECT Name FROM People WHERE DISTINCT;

**✅ Question 2:**

What does this query do?

SELECT Name, Age FROM People ORDER BY Age ASC LIMIT 2;

A) Returns the oldest 2 people  
B) Returns all people ordered by name  
C) Returns the youngest 2 people  
D) Returns 2 people with the highest age

**✅ Question 3:**

Which query renames the Age column to YearsOld?

A) SELECT Age TO YearsOld FROM People;  
B) SELECT Age = YearsOld FROM People;  
C) SELECT Age AS YearsOld FROM People;  
D) SELECT Age FROM People AS YearsOld;

Take your time — let me know your answers when you're ready!

**You said:**

1B 2C 3C

**ChatGPT said:**

🔥 You're on a roll! Let’s review your answers:

**✅ Question 1:**SELECT DISTINCT Name FROM People;

✔ **Your answer: B – Correct!**  
DISTINCT removes duplicate values. ✅

**✅ Question 2:**SELECT Name, Age FROM People ORDER BY Age ASC LIMIT 2;

✔ **Your answer: C – Correct!**  
This returns the **youngest 2 people** (ASC = ascending). ✅

**✅ Question 3:**SELECT Age AS YearsOld FROM People;

✔ **Your answer: C – Correct!**  
This renames the Age column to YearsOld. ✅

✅✅✅ **Perfect score again! You're building solid SQL skills.**

**🟢 Topic 4: Filtering Data (**WHERE**Clause)**

**✅ 1. What is the**WHERE**Clause?**

* The WHERE clause filters records that meet specific conditions.
* It’s used with SELECT, UPDATE, DELETE, etc.

**✅ 2. Basic Syntax**

SELECT \* FROM People WHERE Age > 25;

Returns all rows where Age is greater than 25.

**✅ 3. Comparison Operators**

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| = | Equal to | Age = 30 |
| != or <> | Not equal to | Age != 25 |
| > | Greater than | Age > 20 |
| < | Less than | Age < 40 |
| >= | Greater or equal | Age >= 18 |
| <= | Less or equal | Age <= 30 |

**✅ 4. Logical Operators**

* Combine multiple conditions with AND, OR, NOT.

SELECT \* FROM People WHERE Age > 20 AND Name = 'Alice';

**✅ 5. Special Conditions**

* BETWEEN – between two values (inclusive)

SELECT \* FROM People WHERE Age BETWEEN 20 AND 30;

* IN – matches any value in a list

SELECT \* FROM People WHERE Name IN ('Alice', 'Bob');

* LIKE – pattern matching (used with % and \_)

SELECT \* FROM People WHERE Name LIKE 'A%'; -- Names starting with 'A'

* IS NULL – checks for NULL values

SELECT \* FROM People WHERE Age IS NULL;

**🧪 Example Table:**People

| **ID** | **Name** | **Age** |
| --- | --- | --- |
| 1 | Alice | 25 |
| 2 | Bob | 30 |
| 3 | Carol | 22 |
| 4 | Dave | NULL |

**✅ Example Queries**

-- Find people older than 24 SELECT \* FROM People WHERE Age > 24; -- Find Alice only SELECT \* FROM People WHERE Name = 'Alice'; -- People aged between 20 and 30 SELECT \* FROM People WHERE Age BETWEEN 20 AND 30; -- People named Alice or Bob SELECT \* FROM People WHERE Name IN ('Alice', 'Bob'); -- Names starting with 'C' SELECT \* FROM People WHERE Name LIKE 'C%'; -- Rows where Age is NULL SELECT \* FROM People WHERE Age IS NULL;

**✅ Summary Table**

| **Filter Type** | **SQL Example** |
| --- | --- |
| Equal to | WHERE Age = 30 |
| Not equal to | WHERE Age != 30 |
| Greater than | WHERE Age > 25 |
| Between values | WHERE Age BETWEEN 20 AND 30 |
| In a list | WHERE Name IN ('Alice', 'Bob') |
| Pattern matching | WHERE Name LIKE 'A%' |
| NULL check | WHERE Age IS NULL |

**✅ Question 1:**

Which SQL query returns people whose Age is between 20 and 30 (inclusive)?

A) SELECT \* FROM People WHERE Age > 20 AND Age < 30;  
B) SELECT \* FROM People WHERE Age BETWEEN 20 AND 30;  
C) SELECT \* FROM People WHERE Age >= 20 AND Age <= 30;  
D) Both B and C are correct

**✅ Question 2:**

What does the LIKE operator do in SQL?

A) Filters values that are equal to a given string  
B) Filters values that match a specified pattern  
C) Filters values based on a numeric range  
D) Filters values that are NULL

**✅ Question 3:**

Which of the following would return people whose name starts with 'A'?

A) SELECT \* FROM People WHERE Name LIKE '%A';  
B) SELECT \* FROM People WHERE Name LIKE 'A%';  
C) SELECT \* FROM People WHERE Name = 'A%';  
D) SELECT \* FROM People WHERE Name LIKE '\_A%';

Let me know your answers when you’re ready!

**You said:**

1D 2B 3B

**ChatGPT said:**

Perfect! Let’s review your answers:

**✅ Question 1:**

Which query returns people with Age between 20 and 30 (inclusive)?  
✔ Your answer: **D** — Both BETWEEN 20 AND 30 and Age >= 20 AND Age <= 30 are correct.  
✅ **Correct!**

**✅ Question 2:**

What does the LIKE operator do?  
✔ Your answer: **B** — Matches a specified pattern (with % and \_).  
✅ **Correct!**

**✅ Question 3:**

Which query returns names starting with 'A'?  
✔ Your answer: **B** — LIKE 'A%' means starts with 'A'.  
✅ **Correct!**

✅✅✅ **You crushed it!**

**🟢 Topic 5: Working with Multiple Tables (Joins)**

**✅ 1. What are Joins?**

A **join** combines columns from two (or more) tables based on a **related column**. This is especially useful when your data is split across multiple tables, like customer and order data.

**✅ 2. Types of Joins**

* INNER JOIN: Returns only the rows where there is a match in both tables.
* LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table and matched rows from the right table. If there’s no match, NULL values will appear.
* RIGHT JOIN (or RIGHT OUTER JOIN): Opposite of LEFT JOIN. Returns all rows from the right table and matched rows from the left table.
* FULL OUTER JOIN: Returns all rows when there is a match in one of the tables. If there's no match, the result will include NULL values.

**✅ 3. Basic Syntax for Joins**

INNER JOIN**Example**

SELECT Orders.OrderID, Customers.Name, Orders.Amount FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

**Explanation**:

* This joins Orders and Customers on the CustomerID column.
* Only orders with matching customers are returned.

LEFT JOIN**Example**

SELECT Customers.Name, Orders.OrderID, Orders.Amount FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

**Explanation**:

* This gets all customers, and if they have orders, it also shows those orders. If they don't have orders, OrderID and Amount will be NULL.

**✅ 4. Creating Example Tables**

Let’s imagine we have the following two tables:

**Customers Table:**

| **CustomerID** | **Name** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |
| 3 | Carol |

**Orders Table:**

| **OrderID** | **CustomerID** | **Amount** |
| --- | --- | --- |
| 101 | 1 | 250 |
| 102 | 2 | 100 |
| 103 | 1 | 150 |

**✅ 5. Examples with Data**

INNER JOIN**(Only matching rows)**

SELECT Customers.Name, Orders.OrderID, Orders.Amount FROM Customers INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

Result:

| **Name** | **OrderID** | **Amount** |
| --- | --- | --- |
| Alice | 101 | 250 |
| Bob | 102 | 100 |
| Alice | 103 | 150 |

LEFT JOIN**(All customers, even if they don’t have orders)**

SELECT Customers.Name, Orders.OrderID, Orders.Amount FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

Result:

| **Name** | **OrderID** | **Amount** |
| --- | --- | --- |
| Alice | 101 | 250 |
| Bob | 102 | 100 |
| Alice | 103 | 150 |
| Carol | NULL | NULL |

**✅ 6.**RIGHT JOIN**and**FULL OUTER JOIN

* RIGHT JOIN gives all rows from the right table and matching rows from the left.
* FULL OUTER JOIN returns all rows from both tables, with NULL where there’s no match.

These are more advanced and aren’t as commonly used, but good to know.

**✅ Summary**

| **Join Type** | **Description** | **Example** |
| --- | --- | --- |
| INNER JOIN | Returns only matching rows | SELECT \* FROM Customers INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID; |
| LEFT JOIN | Returns all rows from the left table, and matching rows from the right | SELECT \* FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID; |
| RIGHT JOIN | Returns all rows from the right table, and matching rows from the left | SELECT \* FROM Orders RIGHT JOIN Customers ON Customers.CustomerID = Orders.CustomerID; |
| FULL OUTER JOIN | Returns all rows from both tables, with NULLs for unmatched rows | SELECT \* FROM Customers FULL OUTER JOIN Orders ON Customers.CustomerID = Orders.CustomerID; |

**🔄 Additional Joins & Set Operations in SQL**

**✅ 1. CROSS JOIN (a.k.a. Cartesian Join)**

* **Definition**: Combines every row from the first table with every row from the second table.
* **Use case**: When you want **all possible combinations** of two tables.

**📌 Syntax:**

SELECT \* FROM TableA CROSS JOIN TableB;

**🧪 Example:**

**Customers:**

| **ID** | **Name** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |

**Products:**

| **ID** | **Product** |
| --- | --- |
| 1 | Laptop |
| 2 | Phone |

**Result of**CROSS JOIN**:**

| **Name** | **Product** |
| --- | --- |
| Alice | Laptop |
| Alice | Phone |
| Bob | Laptop |
| Bob | Phone |

⚠️ **Note**: Be careful with large tables — the result size = rows\_in\_A × rows\_in\_B.

**✅ 2. SELF JOIN**

* **Definition**: A table joins to itself.
* Often used to compare rows in the same table (e.g., employee-manager relationship).

**📌 Syntax:**

SELECT A.Name AS Employee, B.Name AS Manager FROM Employees A JOIN Employees B ON A.ManagerID = B.EmployeeID;

**✅ 3. UNION and UNION ALL (Set Operations)**

* **UNION** combines results of two SELECT queries and removes duplicates.
* **UNION ALL** keeps all duplicates.

**📌 Syntax:**

SELECT Name FROM Customers UNION SELECT Name FROM Suppliers;

* ✅ Both SELECT statements must have the same number of columns and compatible types.

**✅ 4. FULL OUTER JOIN**

* Already mentioned, but a quick recap:
* Returns **all rows** from both tables. Where no match is found, NULLs fill in.

**📌 Syntax (PostgreSQL, SQL Server):**

SELECT \* FROM Customers FULL OUTER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

**✅ Summary Table**

| **Join Type / Operation** | **Description** |
| --- | --- |
| INNER JOIN | Only matching rows from both tables |
| LEFT JOIN | All rows from left table + matches from right |
| RIGHT JOIN | All rows from right table + matches from left |
| FULL OUTER JOIN | All rows from both tables |
| CROSS JOIN | All combinations (Cartesian product) |
| SELF JOIN | A table joined with itself |
| UNION | Combines two SELECT results (removes duplicates) |
| UNION ALL | Combines two SELECT results (keeps duplicates) |