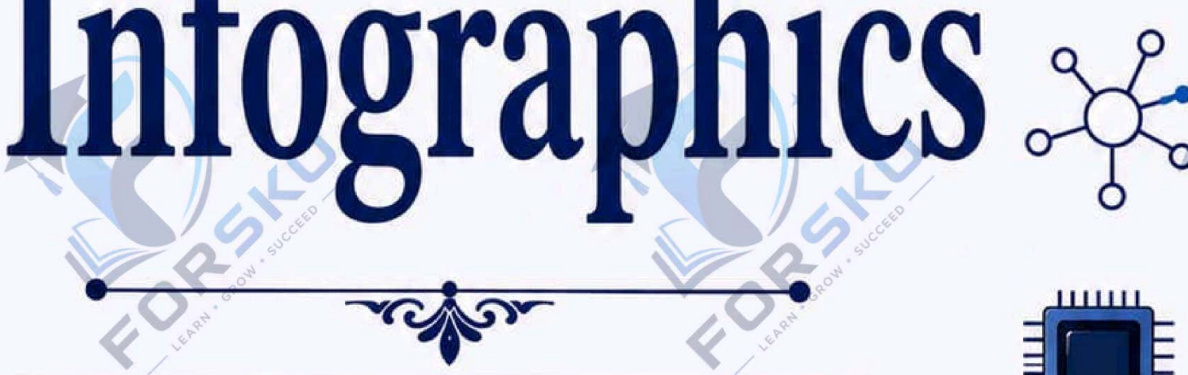
Infographics



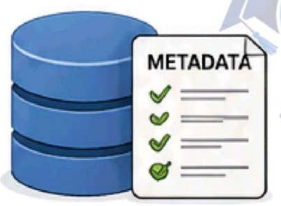

1010101010
0101010101
1010101010
0101010101
</>
</>
Unit - III
10110
01001
10101
DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS
#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!";
return 0;
}

CHARACTERISTICS OF DATABASE

A Database is an organized collection of related data that is stored and managed to meet the needs of multiple users efficiently.

MAIN CHARACTERISTICS

1 SELF-DESCRIBING NATURE

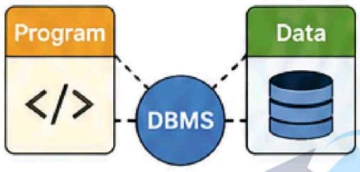


A database contains its own description (metadata) which describes the structure, data types, relationships, and constraints.

2 PROGRAM-DATA INDEPENDENCE

Changes in the data structure do not affect the programs and changes in programs do not affect the data.

3 DATA INDEPENDENCE

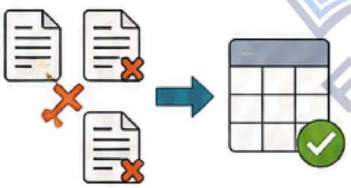


The data can be changed without affecting the application programs that use the data.

4 SHARING OF DATA

Data is shared among multiple users and applications with proper concurrency control.

5 REDUCED DATA REDUNDANCY



Data is stored once and used many times, which reduces duplication and saves storage space.

6 DATA INTEGRITY

Integrity constraints ensure that the data is accurate, consistent and reliable.

7 SECURITY

A database provides security mechanisms to protect data from unauthorized access and misuse.

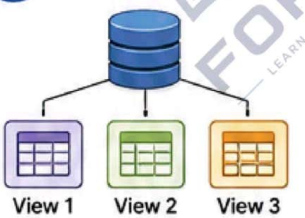
8 CONCURRENCY CONTROL

The database system controls simultaneous access to data by multiple users to avoid conflicts.

9 BACKUP AND RECOVERY

The database system provides backup and recovery facilities to restore data in case of failure.

10 MULTIPLE VIEWS



Different users can have different views of the same data based on their requirements.

SUMMARY

These characteristics make a database system efficient, reliable, secure and easy to use for organizations.

RELATIONAL MODEL

The Relational Model is a way of organizing data in the form of tables (relations) made up of rows and columns. It was proposed by **E.F. Codd in 1970.**

1. WHAT IS RELATIONAL MODEL?

It represents all data in the database as a collection of tables (relations). Each table consists of rows (tuples) and columns (attributes). Relationships between tables are established using keys.

2. BASIC TERMS

- **Relation (Table)** : A table that stores data.
- **Tuple (Row)** : A single row in a table.
- **Attribute (Column)** : A column in a table.
- **Domain** : The set of all possible values for an attribute.
- **Degree** : Number of attributes (columns) in a relation.
- **Cardinality** : Number of tuples (rows) in a relation.

3. STRUCTURE OF A RELATION (TABLE)

STUDENT (A Relation / Table)

RollNo (PK)	Name	Course	Email
101	Arjun	BSc CS	arjun@email.com
102	Neha	BSc IT	neha@email.com
103	Ravi	BSc CS	ravi@email.com
104	Pooja	BSc IT	pooja@email.com

Primary Key (Identifies each row uniquely) — Attributes (Columns) — Tuples (Rows)

4. KEYS IN RELATIONAL MODEL

- **Super Key** : One or more attributes that uniquely identify a tuple.
- **Candidate Key** : Minimal super key (no extra attributes).
- **Primary Key** : One candidate key chosen to identify tuples uniquely.
- **Foreign Key** : Attribute(s) in one table that refer to the primary key of another table.

5. RELATIONSHIP BETWEEN TABLES

STUDENT

RollNo (PK)	Name
101	Arjun
102	Neha
103	Ravi

FK references PK

ENROLLMENT

RollNo (FK)	Course
101	DBMS
102	Python
103	DBMS

ENROLLMENT.RollNo is a Foreign Key (FK) that refers to STUDENT.RollNo which is the Primary Key (PK).

6. FEATURES OF RELATIONAL MODEL

- Data is represented in simple table format.
- Each table has a primary key to uniquely identify rows.
- Relationships are established using keys.
- Reduces data redundancy and inconsistency.
- Supports data integrity and independence.
- Based on mathematical foundation (set theory and logic).

7. EXAMPLE OF RELATIONAL DATABASE

STUDENT

RollNo (PK)	Name	Department
101	Arjun	CS
102	Neha	IT
103	Ravi	CS

1 — N

ENROLLMENT

EnrollID (PK)	RollNo (FK)	CourseID (FK)
1	101	C01
2	102	C02
3	103	C01

N — 1

COURSE

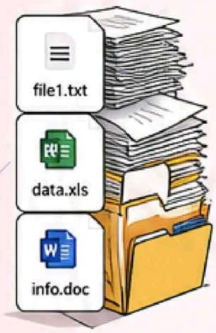
CourseID (PK)	CourseName
C01	DBMS
C02	Python
C03	Data Structures

IN SHORT Relational Model organizes data into related tables using keys and relationships, making data management efficient, consistent and easy to understand.

Need of Database

Why Databases are Essential in Today's World?

Unorganized Files (Traditional Approach)



- ✕ Data is scattered in many files
- ✕ Redundant and duplicate data
- ✕ Inconsistent and unreliable data
- ✕ Difficult to share and access
- ✕ Low security and integrity
- ✕ Hard to maintain and update
- ✕ Time-consuming search

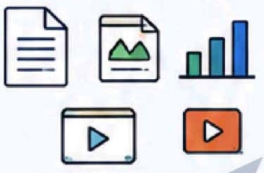
VS

Structured Database (Better Approach)

- ✓ Data is organized in tables
- ✓ Minimizes redundancy
- ✓ Consistent and accurate data
- ✓ Easy data sharing and access
- ✓ High security and integrity
- ✓ Easier maintenance and updates
- ✓ Fast and efficient retrieval

ID	Name	Dept	City
101	Asha	BCA	Pune
102	Rahul	BSA	Delhi
103	Neha	B.Com	Jaipur
...	...	...	...

DATA



Raw facts and figures from various sources

DATABASE MANAGEMENT SYSTEM (DBMS)

Stores, organizes, manages, and processes data efficiently

USEFUL INFORMATION

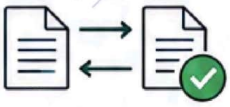
Meaningful, accurate and timely information

10 Key Reasons – Need of Database

1 Organize Large Volumes of Data

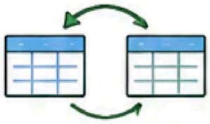
Databases store huge amounts of data in an organized and systematic manner.

2 Reduce Data Redundancy



Eliminates duplicate data and saves storage space and maintenance cost.

3 Improve Data Consistency



Ensures uniform data format and consistent values across the system.

4 Enable Data Sharing

Allows data to be shared across departments, locations and platforms.

5 Provide Data Security

Protects data from unauthorized access, loss, theft or misuse.

6 Support Data Integrity

Enforces accuracy and validity of data using rules and constraints.

7 Allow Fast Data Retrieval

Quickly search, sort and retrieve required data using powerful queries.

8 Offer Backup and Recovery

Regular backup protects data and allows recovery in case of failure.

9 Maintain Data Independence

Changes in data storage or structure do not affect applications or users.

10 Support Multiple Users and Applications

Handles concurrent access by multiple users and different applications.

BSA Semester III – Unit 1: Introduction to Database

SQL and DBMS foundation

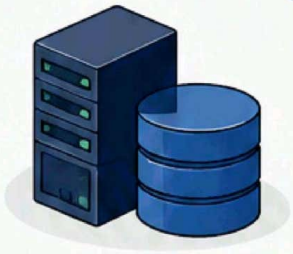
Better Data • Better Decisions • Better Future

ARCHITECTURE OF DBMS

DBMS Architecture defines the structure and components of a Database Management System and how they interact to provide data storage, management and retrieval.

1. Overview

DBMS Architecture shows the various components of a DBMS system and the flow of data between users, queries, and the stored database.



2. Three Level Architecture (ANSI/SPARC)

EXTERNAL LEVEL (VIEW LEVEL)
Multiple user views (external schemas). Each user sees the database as per their requirement.

↔ **External / Conceptual Mapping**
Maps user views to the conceptual level.

CONCEPTUAL LEVEL (LOGICAL LEVEL)
The community view of the database. Describes what data is stored and the relationships among data.

↔ **Conceptual / Internal Mapping**
Maps the conceptual schema to the internal level.

INTERNAL LEVEL (PHYSICAL LEVEL)
Describes how the data is actually stored. Concerned with storage structures and access paths.

↔ **Data Independence**
- Logical Data Independence
- Physical Data Independence

3. Components of DBMS Architecture

Users/Programs
End users or applications interact with the DBMS using queries.

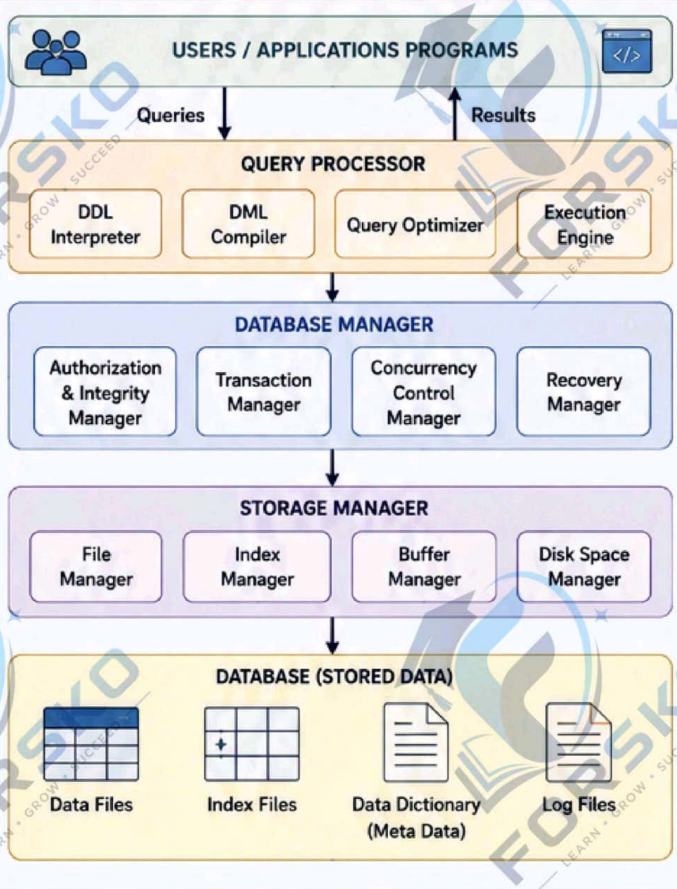
Query Processor
Processes DDL/DML queries, checks syntax and semantics, and generates execution plan.

Database Manager
Coordinates all activities in the system, manages storage structures, transaction and concurrency control.

Storage Manager
Handles the interaction between the low-level data stored in the database and the application programs.

Database
The actual data stored in the form of files, indexes, logs, etc.

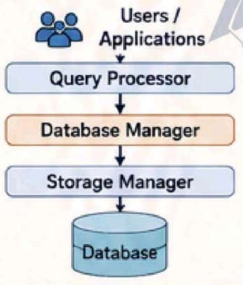
4. Internal Architecture of DBMS



Data Flow

1. Users send queries to DBMS.
2. Query processor interprets and optimizes queries.
3. Database manager manages transactions, security and concurrency.
4. Storage manager accesses data from storage.
5. Results are returned to users.

5. Architecture View



6. Key Points

- Provides data abstraction through three levels.
- Ensures data independence.
- Supports security, integrity and concurrency.
- Optimizes query execution for better performance.
- Manages and organizes data efficiently.

In short: DBMS Architecture ensures that users can access data easily, data is stored securely, and the system works efficiently even with multiple users and large volumes of data.

DBA AND ITS ROLE

A Database Administrator (DBA) is responsible for the overall management, security, performance and availability of the database system to ensure that data is accurate, secure and accessible

1. Who is a DBA?

A DBA is a professional who manages the database environment and ensures that data is stored, organized, secured and available for authorized users.

DBA acts as the guardian of data and the backbone of any organization that relies on databases.

2. Key Responsibilities of a DBA

Database Design
Participates in designing the database structure and defines schema, tables, relationships, constraints.

Installation & Configuration
Installs DBMS Software and configures the system for optimal performance.

Security Management
Creates users, assigns privileges and roles. Ensures data security and access control.

Backup & Recovery
Plans and performs backup regularly and ensures recovery in case of failure or disaster.

Performance Tuning
Monitors and tunes the database for efficient performance and faster query processing.

Storage Management
Manages storage space, data files, indexes and ensures optimal utilization.

Data Integrity
Enforces data accuracy, consistency and integrity using constraints and rules.

Concurrency Control
Manages multiple user access and transactions to ensure data consistency.

3. Skills of a DBA

- Strong knowledge of DBMS (Oracle, MySQL, SQL Server, etc.)
- SQL and query optimization
- Backup and recovery strategies
- Security and user management
- Problem solving and troubleshooting
- Performance monitoring and tuning
- Communication and documentation

4. DBA Role in the Organization

Ensures Data Availability
Keeps the database up and running for users.

Ensures Data Security
Protects data from unauthorized access, loss or corruption.

Improves Performance
Optimizes database performance for faster access and processing.

Supports Business Decisions
Provides reliable and accurate data for analysis and reporting.

Ensures Business Continuity
Plans for backup and recovery to avoid data loss and downtime.

5. DBA Works With

End Users	Developers	System Administrators	Management	Auditors
To understand data requirements and provide access.	To design database objects and improve application performance.	For hardware, OS and network related configurations.	To support reporting, analytics and decision making.	To ensure compliance with standards and regulations.

6. Types of DBA

- **System DBA** – Handles overall database system and infrastructure.
- **Application DBA** – Works with application databases and supports developers.
- **Security DBA** – Focuses on database security and user access control.
- **Data Warehouse DBA** – Manages data warehouses and large volumes of data.

7. Importance of DBA

Protects valuable data assets	Ensures high performance and efficiency	Minimizes data loss and system downtime	Supports smooth operations and business growth	Ensures compliance with policies and regulations

> A good DBA ensures that the right data is available to the right people at the right time in the right way.

In Short: DBA plans, manages, secures and optimizes the database system so that organizations can store data safely and use it effectively.

STRUCTURED QUERY LANGUAGE (SQL) INTRODUCTION

SELECT *
FROM Students;

SQL (Structured Query Language) is a standard programming language used to **create, manage, manipulate and retrieve data** from relational databases.

1. What is SQL?

- SQL is a non-procedural (declarative) language.
- It is used to communicate with databases.
- SQL follows a simple English-like syntax, making it easy to learn and use.
- SQL is an ANSI (American National Standards Institute) standard language.

2. Purpose of SQL

- Create and modify database structures.
- Insert, update, delete and retrieve data.
- Control access to data.
- Maintain data integrity and security.
- Perform transactions and manage concurrency.

3. Key Features of SQL

Standard Language
SQL is a standardized language for database systems.

Easy to Learn
Uses simple, English-like commands.

Powerful and Flexible
Supports complex queries and data operations.

Data Independence
Separates how data is stored from how it is used.

Security
Provides built-in security and access control.

Set-Based Language
Works on sets of data, not individual records.

4. SQL Works with Relational Databases

SQL works with data stored in tables (relations) which are made up of rows and columns.

Example Table: Students

ID	Name	Course	City
101	Asha	BCA	Pune
102	Rahul	BSA	Delhi
103	Neha	B.Com	Jaipur

Column (Attribute)

Row (Tuple)

SQL helps us perform operations on this data such as retrieving, inserting, updating and deleting.

5. Why is SQL Important?

It is widely used in various applications like banking, e-commerce, education, healthcare, etc.

It is supported by almost all database systems (MySQL, Oracle, SQL Server, PostgreSQL, etc.).

It is essential for data analysts, developers, DBAs and IT professionals.

6. Example: Simple SQL Queries

Query	Description
SELECT * FROM Students;	Retrieves all records from Students table.
SELECT Name, City FROM Students;	Retrieves Name and City columns.
SELECT * FROM Students WHERE City = 'Pune';	Retrieves students from Pune.
INSERT INTO Students (ID, Name, Course, City) VALUES (104, 'Karan', 'BCA', 'Mumbai');	Inserts a new record.
UPDATE Students SET City = 'Hyderabad' WHERE ID = 101;	Updates the city of student with ID 101.
DELETE FROM Students WHERE ID = 103;	Deletes the record with ID 103.

7. SQL at a Glance

User
User writes SQL Query

→

SQL Query
Query is sent to the database

→

Database Engine
Processes the query and accesses data

→

Result Set
Data is retrieved as result

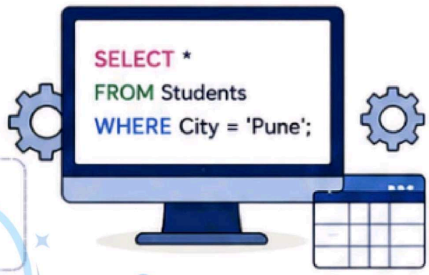
→

Output
Results are displayed to the user

★ In Short

SQL is the foundation for managing and interacting with relational databases. It allows us to store, access, manipulate and secure data efficiently.

SQL FEATURES & CHARACTERISTICS



SQL (Structured Query Language) is a powerful, standard and user-friendly language used to create, access and manage data in relational databases.

SQL FEATURES

1 Standard Language
SQL is an ANSI/ISO standard language (ISO/IEC 9075). It is widely accepted and used across different DBMS.

2 Easy to Learn and Use
SQL has simple English-like syntax and keywords, making it easy to learn and understand.

3 Powerful and Flexible
Supports a wide range of operations such as data retrieval, insertion, update, delete, and complex queries.

4 Data Manipulation
Allows users to insert, update, delete and retrieve data efficiently from one or more tables.

5 Data Security
Provides access control, user authentication and privileges to protect data from unauthorized access.

6 Support for Relationships
Easily handles relationships between tables using keys (Primary Key, Foreign Key) and joins.

7 High Performance
Optimized query processing and indexing improve the speed and efficiency of data operations.

8 Scalability
SQL is suitable for small applications as well as large enterprise-level database systems.

SQL CHARACTERISTICS

1 Declarative (Non-Procedural)
SQL is declarative, not procedural. We tell the system what data we want, not how to get it.

2 Set-Oriented Language
SQL works on sets of data (rows), not on individual records.

3 Used for Relational Databases
SQL is specifically designed for relational database management systems (RDBMS).

4 High-Level Language
SQL is a high-level language. It hides the internal storage details from the user.

5 Data Independence
Provides both Logical Data Independence and Physical Data Independence.

6 Portable
SQL queries are portable across different database systems with little or no modification.

7 Integrity Constraints
SQL supports constraints like PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL to ensure data accuracy.

8 Transaction Management
SQL supports ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure reliable transactions.

9 Multi-User Support
SQL allows multiple users to access and work on the database simultaneously.

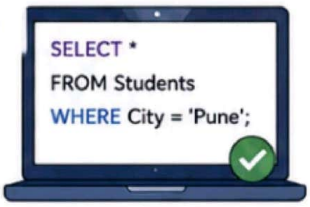
10 Extensible
SQL can be extended with additional features, functions and procedures to meet application needs.

In Short

SQL is a standard, powerful, flexible and easy-to-use language with strong features and characteristics that make it the ideal choice for managing relational databases efficiently and securely.

SQL ADVANTAGES

SQL (Structured Query Language) provides an efficient and standardized way to manage and interact with data in relational database systems.



1 Easy to Learn and Use

SQL has a simple English-like syntax that is easy to learn for beginners and powerful enough for advanced users.

2 Standard Language

SQL is an ANSI/ISO standard (ISO/IEC 9075) used by almost all relational database systems, ensuring compatibility.

3 Works with Relational Databases

Designed specifically for relational databases, SQL efficiently manages data stored in tables and the relationships between them.

4 High Performance

SQL queries are optimized by the database engine for fast data retrieval and manipulation, improving application performance.

5 Supports Multiple Users

Allows multiple users to access and work on data simultaneously while maintaining data integrity.

6 Data Integrity and Security

SQL provides strong security features and supports constraints (primary key, foreign key, unique, check, not null) to ensure accurate and consistent data.

7 Data Independence

Changes in the database structure do not affect the application programs (logical and physical data independence is supported).

8 Flexibility

SQL supports a wide range of operations (data definition, manipulation, control) and can handle simple to complex queries easily.

9 Scalability

SQL is suitable for small applications as well as large enterprise-level database systems.

10 Integration

SQL can be easily integrated with various programming languages, tools and applications.

SQL DATA TYPES

SQL data types define the type of data that can be stored in a column of a table.

Category	Data Types	Description	Example
Numeric Types	INT / INTEGER, SMALLINT, BIGINT DECIMAL(p,s) / NUMERIC(p,s) FLOAT(p), REAL, DOUBLE PRECISION	Store whole numbers, decimal numbers and approximate floating-point numbers.	101, -50, 1000 12.50, 999.99 3.14, 2.71828
Character/ String Types	CHAR(n), VARCHAR(n), TEXT / CLOB	Store fixed-length, variable-length or large text data.	'A', 'SQL' 'John Doe', 'Pune' 'This is a long text....'
Date & Time Types	DATE, TIME, TIMESTAMP, DATETIME, YEAR	Store date, time or date-time values.	'2024-05-20' '14:30:00' '2024-05-20 14:30:00'
Boolean Type	BOOLEAN	Store true/false values.	TRUE, FALSE
Binary Types	BINARY(n), VARBINARY(n), BLOB	Store binary data such as images, audio, files.	X'ABCD' (Binary data)
Other Types	ENUM, SET, JSON, XML	Store predefined values, multiple values, JSON or XML data.	ENUM('Male','Female') JSON object XML document

Key Points

- Choosing the right data type improves storage efficiency and performance.
- Some data types may vary slightly across different DBMS (MySQL, Oracle, SQL Server, PostgreSQL, etc.).
- Always use appropriate size (n, p, s) to manage memory effectively.

SQL provides a rich set of data types to handle different kinds of data accurately and efficiently.

SQL DATA TYPES & OPERATORS

SQL (Structured Query Language) is used to create, manage and retrieve data from relational databases.

```
SELECT *
FROM Students
WHERE City = 'Pune';
```

SQL DATA TYPES

SQL data types define the kind of data that can be stored in a column of a table.

Category	Data Types	Description	Example	Storage / Range
1. Numeric Types	INT / INTEGER SMALLINT BIGINT DECIMAL(p,s) NUMERIC(p,s) FLOAT(p) REAL DOUBLE PRECISION	Store whole numbers, decimal numbers and approximate floating-point numbers.	101 -50 12345 12.50 3.1415	INT: -2,147,483,648 to 2,147,483,647 DECIMAL(5,2): -999.99 to 999.99 FLOAT: Approximate numeric values
2. Character / String Types	CHAR(n) VARCHAR(n) TEXT CLOB	Store fixed-length or variable-length character data.	'A' 'John Doe' 'Hello World'	CHAR(n): Fixed length VARCHAR(n): 0 to n characters TEXT/CLOB: Large text data
3. Date & Time Types	DATE TIME TIMESTAMP DATETIME YEAR	Store date, time or date-time values.	'2024-05-20' '14:30:00' '2024-05-20 14:30:00' '2024-05-20 14:30:00' '2024'	DATE: YYYY-MM-DD TIME: HH:MI:SS TIMESTAMP: Date + Time YEAR: YYYY
4. Boolean Type	BOOLEAN	Store true/false values.	TRUE FALSE	TRUE or FALSE (1 or 0 in some DBMS)
5. Binary Types	BINARY(n) VARBINARY(n) BLOB	Store binary data such as images, audio, files.	X'1A2B' (Binary data)	Up to n bytes or large binary objects (BLOB)
6. Other / Advanced Types	ENUM SET JSON XML	Store predefined values, multiple values, JSON or XML data.	ENUM('Male', 'Female') SET('A','B','C') {"name":"Asha"} <note>Data</note>	DBMS specific support (MySQL, PostgreSQL, Oracle, SQL Server, etc.)

Note: The exact size and range may vary slightly between different DBMS (MySQL, Oracle, SQL Server, PostgreSQL, etc.). Always use appropriate data types to save storage and ensure data integrity.

SQL OPERATORS

Operators are special symbols or keywords used to perform operations on data and values in SQL.

1. Arithmetic Operators

Operator	Meaning	Example	Result
+	Addition	10 + 5	15
-	Subtraction	10 - 5	5
*	Multiplication	10 * 5	50
/	Division	10 / 5	2
%	Modulus (Remainder)	10 % 3	1

2. Comparison Operators

Operator	Meaning	Example	Result
=	Equal to	A = B	True/False
<> or !=	Not equal to	A <> B	True/False
>	Greater than	A > B	True/False
<	Less than	A < B	True/False
>=	Greater than or equal to	A >= B	True/False
<=	Less than or equal to	A <= B	True/False

3. Logical Operators

Operator	Meaning	Example
AND	True if both conditions are true	A > 10 AND B < 20
OR	True if any one condition is true	A > 10 OR B < 20
NOT	Negates the condition	NOT (A > 10)

4. Pattern Matching Operators

Operator	Meaning	Example	Matches
LIKE	Matches a pattern	'A%'	Starts with 'A'
_ (underscore)	Matches any single character	'A_B'	A followed by any one char then B
% (percent)	Matches any sequence of chars	'%xyz%'	Contains 'xyz' anywhere

5. IN / Between / IS Operators

Operator	Meaning	Example
IN	Matches any value in a list	City IN ('Pune', 'Delhi')
NOT IN	Not matches in a list	City NOT IN ('Pune')
BETWEEN	Between a range	Age BETWEEN 18 AND 25
NOT BETWEEN	Not between a range	Age NOT BETWEEN 18 AND 25
IS NULL	Checks for NULL	Address IS NULL
IS NOT NULL	Checks for NOT NULL	Address IS NOT NULL

6. Bitwise Operators

Operator	Meaning	Example
&	Bitwise AND	A & B
|	Bitwise OR	A | B
^	Bitwise XOR	A ^ B
~	Bitwise NOT	~A
<<	Left shift	A << 2
>>	Right shift	A >> 2

7. Assignment Operators

Operator	Meaning	
=	Assign	@x = 10
+=	Add and assign	@x += 5 (Equivalent to @x = @x + 5)
-=	Subtract and assign	@x -= 5 (Equivalent to @x = @x - 5)
*=	Multiply and assign	@x *= 5 (Equivalent to @x = @x * 5)
/=	Divide and assign	@x /= 5 (Equivalent to @x = @x / 5)
%=	Modulus and assign	@x %= 5 (Equivalent to @x = @x % 5)

8. Other Operators

Operator	Meaning	Example
+ (Concatenation)	Concatenate strings	'John' + ' Doe'
()	Group expressions	(A + B) * C
,	Separator (in lists)	SELECT id, name FROM Students
EXISTS	Checks existence of rows	WHERE EXISTS (SELECT 1 FROM Orders)
ANY / ALL	Compare with any or all values	price > ANY (SELECT price FROM Products)

In Short

SQL Data Types define what kind of data we can store.
SQL Operators help us perform calculations, comparisons, and logical operations on that data.

COMPONENTS OF SQL

SQL is divided into different components (sub-languages) based on the type of operations they perform.

```
SELECT *
FROM Students
WHERE City = 'Pune';
```

1. DDL (Data Definition Language)

Used to define and manage the structure of database objects.

Example Commands: CREATE, ALTER, DROP, TRUNCATE, RENAME

2. DML (Data Manipulation Language)

Used to retrieve and manipulate the data in database objects.

Example Commands: SELECT, INSERT, UPDATE, DELETE

3. DCL (Data Control Language)

Used to control access and permissions on data in the database.

Example Commands: GRANT, REVOKE

4. TCL (Transaction Control Language)

Used to manage transactions in the database.

Example Commands: COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION

DDL (Data Definition Language)

DDL commands are used to define, modify and manage the database structure (schemas, tables, indexes, constraints, etc.). These changes affect the structure, not the data.

Common DDL Commands

Command	Purpose	Example	Description
CREATE	Creates a new database object (database, table, view, index, etc.).	`CREATE TABLE Students ( ID INT PRIMARY KEY, Name VARCHAR(50), City VARCHAR(30) );`	Creates a new table named 'Students'.
ALTER	Modifies the structure of an existing database object.	`ALTER TABLE Students ADD Email VARCHAR(50);`	Adds a new column 'Email' in existing table 'Students'.
DROP	Deletes an existing database object completely.	`DROP TABLE Students;`	Deletes the table 'Students' and its structure.
TRUNCATE	Removes all records from a table, but keeps the structure intact.	`TRUNCATE TABLE Students;`	Deletes all rows from the table 'Students' but table structure remains.
RENAME	Renames an existing database object.	`RENAME TABLE Students TO Student_Info;`	Renames table 'Students' to 'Student_Info'.

Key Points

- DDL commands work on the structure (schema) of the database.
- Changes made by DDL commands are auto committed.
- DDL defines how data is stored and organized in the database.
- Only authorized users can execute DDL commands.
- Examples of objects managed by DDL: Tables, Views, Indexes, Schemas, Constraints.

Example: DDL Workflow

1. **CREATE** — `CREATE TABLE Students (....);`
2. **ALTER** — `ALTER TABLE Students ADD Email VARCHAR(50);`
3. **DROP** — `DROP TABLE Students;`
4. **TRUNCATE** — `TRUNCATE TABLE Students;`
5. **RENAME** — `RENAME TABLE Students TO Student_Info;`

In Short

DDL commands are the foundation of any database.
They define the structure that stores and organizes the data.

COMPONENTS OF SQL – DML
(DATA MANIPULATION LANGUAGE)

```
SELECT *
FROM Students
WHERE City = 'Pune';
```

DML commands are used to retrieve, insert, update and delete data in database tables. These commands work with the data stored in the database.

DML COMPONENTS

1 SELECT

Retrieves data from one or more tables.

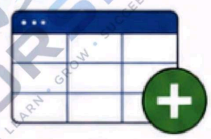
Syntax

```
SELECT column1, column2, ...
FROM table_name
[WHERE condition]
[ORDER BY column]
[GROUP BY column]
[HAVING condition];
```

Example

```
SELECT Name, City
FROM Students
WHERE City = 'Pune';
```

2 INSERT



Inserts new records into a table.

Syntax

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

Example

```
INSERT INTO Students
(ID, Name, City, Course)
VALUES (101, 'Asha', 'Pune',
'BCA');
```

3 UPDATE

Modifies existing records in a table.

Syntax

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

Example

```
UPDATE Students
SET City = 'Mumbai'
WHERE ID = 101;
```

4 DELETE

Deletes existing records from a table.

Syntax

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

Example

```
DELETE FROM Students
WHERE ID = 101;
```

EXAMPLE TABLE: Students

ID	Name	City	Course	Age
101	Asha	Pune	BCA	20
102	Rahul	Delhi	BSA	21
103	Neha	Mumbai	B.Com	19

DML IN ACTION

- **SELECT** – Retrieves data → View or Report
- **INSERT** – Adds new data → New record created
- **UPDATE** – Modifies data → Existing record updated
- **DELETE** – Removes data → Record deleted

KEY POINTS

- DML works on the data (rows) of tables.
- It does not change the structure of the table.
- Changes made by DML commands are temporary until saved using TCL (COMMIT / ROLLBACK).
- Proper WHERE clause is important in UPDATE and DELETE to avoid affecting all rows.

DML VS DDL

Aspect	DML	DDL
Purpose	Manipulate data	Define structure
Affects	Data (Rows)	Structure (Tables, Schema)
Examples	SELECT, INSERT, UPDATE, DELETE	CREATE, ALTER, DROP, TRUNCATE
Auto Commit	No (Requires COMMIT)	Yes (Auto Commit)

In Short

DML is the heart of data handling in SQL. It allows users to retrieve, add, modify and remove data efficiently from database tables.

COMPONENTS OF SQL – DCL
(DATA CONTROL LANGUAGE)

```
SELECT *
FROM Students
WHERE City = 'Pune';
```

DCL commands are used to control access and permissions on database objects. They help keep data secure by allowing only authorized users to access or perform specific actions.

DCL COMPONENTS

SQL has two main DCL commands.

1 GRANT

GRANT is used to give privileges (permissions) to users or roles on database objects.

Syntax

```
GRANT privilege_list
 ON object_name
 TO user_name [, user_name] ...
 [WITH GRANT OPTION];
```

Example

```
GRANT SELECT, INSERT ON Students
TO user1;

GRANT ALL PRIVILEGES ON *.*
TO user2 WITH GRANT OPTION;
```

Common Privileges

SELECT	INSERT	UPDATE	DELETE	EXECUTE
Read data	Add data	Modify data	Remove data	Run procedures

2 REVOKE

REVOKE is used to remove privileges that were previously granted to users or roles.

Syntax

```
REVOKE privilege_list
 ON object_name
 FROM user_name [, user_name] ...
 [CASCADE | RESTRICT];
```

Example

```
REVOKE INSERT ON Students
FROM user1;

REVOKE ALL PRIVILEGES ON *.*
FROM user2 CASCADE;
```

Notes

- CASCADE – Removes the privilege and all rights granted by that user to others.
- RESTRICT – Does not allow revoke if the privilege has been granted to other users.

OBJECTS ON WHICH PRIVILEGES CAN BE APPLIED

Tables | Views | Databases | Stored Procedures | Functions | Sequences

EXAMPLE: DCL IN ACTION

ADMIN

1. **GRANT** – Admin grants SELECT privilege on Students table to user1.

```
GRANT SELECT ON Students
TO user1;
```

2. **USER1 ACCESS** – user1 can now view data from Students table.

```
SELECT *
FROM Students;
```

3. **REVOKE** – Admin revokes SELECT privilege from user1.

```
REVOKE SELECT ON Students
FROM user1;
```

USER1

4. **ACCESS REMOVED** – user1 can no longer view data from Students table.

```
SELECT * FROM Students;
-- Error: You do not have
   permission to execute this
   operation.
```

KEY POINTS

- DCL is used for security and access control.
- GRANT gives privileges to users.
- REVOKE removes privileges from users.
- Privileges can be object specific (e.g., a table) or global.
- Only users with appropriate privileges (e.g., ADMIN) can grant or revoke access.
- Helps enforce data security and maintain integrity.

DCL SUMMARY

Command	Purpose	Example	Effect
GRANT	Gives privileges to users	GRANT SELECT ON Students TO user1;	user1 can access the table
REVOKE	Removes privileges from users	REVOKE SELECT ON Students FROM user1;	user1 can no longer access the table

In Short

DCL commands (GRANT, REVOKE) are essential for controlling who can access what in a database. They ensure that only authorized users can perform actions on database objects, keeping your data safe and secure.

COMPONENTS OF SQL - TCL
(TRANSACTION CONTROL LANGUAGE)

```
SELECT *
FROM Students
WHERE City = 'Pune';
```

TCL commands are used to manage transactions in a database. They ensure data integrity by allowing you to save, undo, or manage a group of SQL operations as a single unit of work.

TCL COMPONENTS

SQL has four main TCL commands.

1 COMMIT

Saves all changes made in the current transaction permanently.

Syntax

```
COMMIT;
```

Example

```
UPDATE Students
SET City = 'Pune'
WHERE ID = 101;
COMMIT;
```

2 ROLLBACK

Undoes all changes made in the current transaction and restores the database to the last committed state.

Syntax

```
ROLLBACK;
```

Example

```
UPDATE Students
SET City = 'Pune'
WHERE ID = 101;
ROLLBACK;  -- Changes undone
```

3 SAVEPOINT

Sets a point within a transaction to which you can later ROLLBACK.

Syntax

```
SAVEPOINT savepoint_name;
```

Example

```
SAVEPOINT sp1;
UPDATE Students SET City = 'Pune'
WHERE ID = 101;
ROLLBACK TO sp1;
```

4 ROLLBACK TO SAVEPOINT

Rolls back the transaction to a specific SAVEPOINT, without affecting earlier changes.

Syntax

```
ROLLBACK TO savepoint_name;
```

Example

```
SAVEPOINT sp1;
UPDATE Students SET City = 'Pune'
WHERE ID = 101;
ROLLBACK TO sp1;
```

HOW TRANSACTIONS WORK

1. Start Transaction
A series of SQL statements are executed.

→

2. Perform Operations
INSERT / UPDATE / DELETE operations are performed.

→

3. Decide
Choose to COMMIT or ROLLBACK.

→

4A. COMMIT
Changes are saved permanently.

OR

4B. ROLLBACK
Changes are discarded (not saved).

EXAMPLE SCENARIO

Transfer ₹500 from Account A to Account B.
(Two update operations must succeed together.)

Step	Action	Description	Account A	Account B
1	Initial State	Before transaction	₹1000	₹500
2	UPDATE A	Debit ₹500 from A	₹500	₹500
3	UPDATE B	Credit ₹500 to B	₹500	₹1000
4A	COMMIT	All changes saved	₹500	₹1000
4B	ROLLBACK	All changes undone	₹1000	₹500

Using TCL ensures either both operations succeed (COMMIT) or both are undone (ROLLBACK), maintaining data integrity.

KEY POINTS

- A transaction is a group of one or more SQL operations treated as a single unit of work.
- COMMIT makes changes permanent.
- ROLLBACK undoes all changes since the last COMMIT.
- SAVEPOINT allows partial rollback within a transaction.
- TCL commands help maintain ACID properties (Atomicity & Consistency) in a database.
- By default, most databases use Auto-Commit Mode (each statement is committed automatically).

TCL COMMANDS SUMMARY

Command	Purpose	Scope	Effect	Example
COMMIT	Save all changes in the transaction	Current Transaction	Makes all changes permanent	`COMMIT;`
ROLLBACK	Undo all changes in the transaction	Current Transaction	Discards all uncommitted changes	`ROLLBACK;`
SAVEPOINT	Set a point within a transaction	Current Transaction	Marks a point to which you can rollback	`SAVEPOINT sp1;`
ROLLBACK TO SAVEPOINT	Undo changes back to a savepoint	From Savepoint to Current	Keeps changes before the savepoint intact	`ROLLBACK TO sp1;`

IN SHORT

TCL commands (COMMIT, ROLLBACK, SAVEPOINT, ROLLBACK TO SAVEPOINT) manage transactions to ensure reliable and consistent data.

SQL SELECT STATEMENT
with Clauses

```
SELECT *
FROM Students
WHERE City = 'Pune';
```

The SELECT statement is used to retrieve data from one or more tables.
Clauses help filter, group, and sort the data as needed.

1 WHERE

Filters rows based on a specified condition.

Syntax

```
SELECT column_list
FROM table_name
WHERE condition;
```

Example

```
SELECT Name, City
FROM Students
WHERE City = 'Pune';
```

2 GROUP BY

Groups rows that have the same values in specified column(s).

Syntax

```
SELECT column_list
FROM table_name
GROUP BY column_name;
```

Example

```
SELECT City, COUNT(*) AS Total
FROM Students
GROUP BY City;
```

3 HAVING

Filters groups based on a condition.

Syntax

```
SELECT column_list
FROM table_name
GROUP BY column_name
HAVING condition;
```

Example

```
SELECT City, COUNT(*) AS Total
FROM Students
GROUP BY City
HAVING COUNT(*) > 1;
```

4 ORDER BY

Sorts the result set in ascending (ASC) or descending (DESC) order.

Syntax

```
SELECT column_list
FROM table_name
ORDER BY column_name [ASC | DESC];
```

Example

```
SELECT Name, Marks
FROM Students
ORDER BY Marks DESC;
```

SAMPLE TABLE: Students

ID	Name	City	Course	Marks
1	Asha	Pune	BCA	85
2	Rahul	Delhi	BCA	78
3	Neha	Pune	BBA	92
4	Rohan	Mumbai	BCA	87
5	Priya	Pune	BCA	90
6	Karan	Delhi	BBA	76
7	Sneha	Mumbai	BBA	88
8	Mohit	Pune	BCA	65

USING ALL CLAUSES TOGETHER

You can combine clauses in the following logical order:

WHERE → GROUP BY → HAVING → ORDER BY

```
SELECT City, COUNT(*) AS Total_Students, AVG(Marks) AS Avg_Marks
FROM Students
WHERE Course = 'BCA'
GROUP BY City
HAVING COUNT(*) > 1
ORDER BY Avg_Marks DESC;
```

Result:

City	Total_Students	Avg_Marks
Pune	3	80.00
Mumbai	1	87.00

Explanation:

- Get students of BCA course,
  group by City,
  keep groups having more than 1 student, and
  sort by average marks in descending order.

ORDER OF CLAUSE APPLICATION

1. **WHERE** — Filters individual rows.
2. **GROUP BY** — Groups the filtered rows.
3. **HAVING** — Filters the groups.
4. **ORDER BY** — Sorts the final result.

→ Final Result

QUICK REFERENCE

Clause	Purpose	Can Use Aggregate Functions?	Filters
WHERE	Filters rows before grouping	No	Row level
GROUP BY	Groups rows with same values	Yes	Groups rows
HAVING	Filters groups after grouping	Yes	Group level
ORDER BY	Sorts the final output	No	Final result

KEY POINTS

- WHERE filters rows before grouping.
- GROUP BY groups rows.
- HAVING filters groups after grouping.
- ORDER BY sorts the final result set.
- The order of clauses (WHERE → GROUP BY → HAVING → ORDER BY) is important and must be followed.
- Aggregate functions: COUNT(), SUM(), AVG(), MIN(), MAX() are used with GROUP BY and HAVING.

In Short

SELECT + clauses help you retrieve the right data from your database.
WHERE filters rows, GROUP BY groups them, HAVING filters groups,
and ORDER BY sorts the final result.

DATA AND INFORMATION

From Raw Facts to Meaningful Knowledge

DATA

Raw Facts and Figures

Data is raw, unprocessed facts and figures that have no specific meaning on their own.

EXAMPLES

- 25, 30, 28, 27, 26
- 101, 102, 103, 104
- 01/05/2025, 02/05/2025
- 500, 1500, 2000, 2500

KEY POINTS

- Raw and unorganized
- Without context or meaning
- Large in volume
- Input to processing
- Stored in databases

ANALOGY

Data is like unbaked ingredients (raw rice).

PROCESSING
(Organize, Sort, (Calculate, Analyze)

INFORMATION

Processed and Meaningful

Information is data that has been processed, organized and given meaning to be useful.

EXAMPLES

- Average temperature in May is 27°C
- Total number of students: 4
- Report generated on 02/05/2025
- Total sales amount: ₹5750

KEY POINTS

- Processed and organized
- Has context and meaning
- Useful for decision making
- Output of processing
- Helps in knowledge generation

ANALOGY

Information is like cooked and ready to eat meal.

KEY DIFFERENCE AT A GLANCE

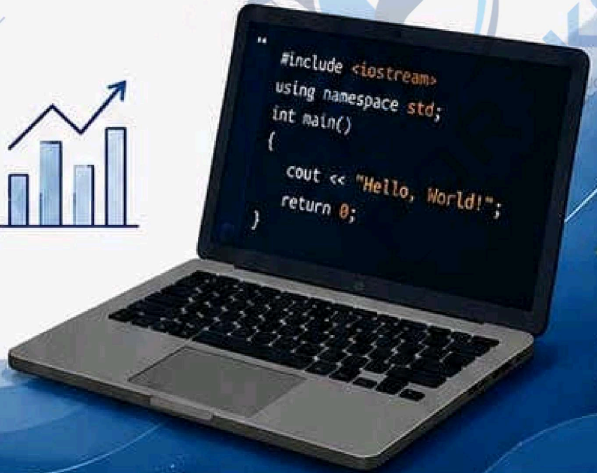
ASPECT	DATA	INFORMATION
Nature	Raw, unprocessed	Processed, meaningful
Meaning	No meaning by itself	Has meaning and context
Use	Input for processing	Used for decision making
Example	25, 30, 28	Average temperature is 27°C
Result	Does not reduce uncertainty	Reduces uncertainty and adds value

SUMMARY

Data is the raw material, while **Information** is the final product. Good information leads to better decisions.

Unit - IV


DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS


#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!";
return 0;
}

RDBMS CONSTRAINTS

Rules that ensure the **accuracy**, **reliability** and **integrity** of data in a database.

TYPES OF CONSTRAINTS

1 PRIMARY KEY

Uniquely identifies each row in a table.
No NULL values allowed.

Example

Each StudentID is unique. No two students can have the same StudentID.

SQL Example

```
StudentID INT
  PRIMARY KEY
```

2 UNIQUE

Ensures all values in a column or set of columns are unique.
Allows NULL values.

Example

Email must be unique for every user.

SQL Example

```
Email VARCHAR(100)
UNIQUE
```

3 NOT NULL

Ensures a column cannot have a NULL value.

Example

Name must be provided. It cannot be left empty.

SQL Example

```
Name VARCHAR(50)
NOT NULL
```

4 CHECK

Ensures that all values in a column satisfy a specific condition.

Example

Age must be greater than or equal to 18.

SQL Example

```
Age INT
CHECK (Age >= 18)
```

5 FOREIGN KEY

Ensures referential integrity by linking one table to another.

Example

DeptID in Employee table must exist in Department table.

SQL Example

```
DeptID INT
FOREIGN KEY (DeptID)
REFERENCES Department(DeptID)
```

6 DEFAULT

Sets a default value for a column when no value is specified.

Example

Status will be 'Active' by default.

SQL Example

```
Status VARCHAR(20)
DEFAULT 'Active'
```

WHY CONSTRAINTS MATTER?

- Maintain data accuracy and consistency
- Prevent invalid or duplicate data
- Enforce business rules
- Ensure referential integrity between tables

REFERENTIAL INTEGRITY (FOREIGN KEY)

Department

DeptID	DeptName
1	IT
2	HR
3	Sales

Employee

EmpID	EmpName	DeptID
101	Asha	1
102	Ravi	2
103	Neha	1

☆ Together, constraints help build a **strong** and **reliable** database.

DATA CONSTRAINTS
AND ITS TYPES

Data Constraints are rules applied to table columns to ensure the accuracy, consistency and integrity of data in a database.

WHY WE USE CONSTRAINTS?

- Ensure data accuracy
- Prevent invalid or duplicate data
- Maintain consistency across the database
- Enforce business rules

TYPES OF DATA CONSTRAINTS

1 PRIMARY KEY

Uniquely identifies each row in a table.
No NULL values allowed.

Example

StudentID	Name	Age
1	Asha	20
2	Ravi	21
3	Neha	22

SQL Example

```
StudentID INT
PRIMARY KEY
```

2 UNIQUE

Ensures all values in a column or set of columns are unique.
Allows NULL values.

Example

Email
asha@email.com
ravi@email.com
neha@email.com

SQL Example

```
Email VARCHAR(100)
UNIQUE
```

3 NOT NULL

Ensures a column cannot have a NULL value.

Example

Name
Asha
Ravi
Neha

SQL Example

```
Name VARCHAR(50)
NOT NULL
```

4 CHECK

Ensures that all values in a column satisfy a specific condition.

Example

Age
21
18
25

SQL Example

```
Age INT
CHECK (Age >= 18)
```

5 FOREIGN KEY

Ensures referential integrity by linking one table to another.

Example

Department

DeptID	DeptName
1	IT
2	HR
3	Sales

Employee

EmpID	EmpName	DeptID
101	Asha	1
102	Ravi	1
103	Neha	2

SQL Example

```
DeptID INT
FOREIGN KEY (DeptID)
REFERENCES Department(DeptID)
```

6 DEFAULT

Sets a default value for a column when no value is specified.

Example

Status
Active
Active
Inactive

SQL Example

```
Status VARCHAR(20)
DEFAULT 'Active'
```

7 COMPOSITE KEY

A primary key that consists of two or more columns.

Example

OrderID	ProductID	Quantity
101	1	2
101	2	1
102	1	3

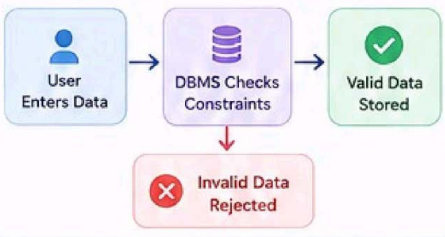
SQL Example

```
PRIMARY KEY (OrderID, ProductID)
```

KEY POINTS

- Constraints help in maintaining valid and reliable data.
- They reduce errors and improve database performance.
- Constraints are enforced by the DBMS when data is inserted or updated.

HOW CONSTRAINTS WORK?



SUMMARY TABLE

Constraint	Ensures	Allows NULL?
Primary Key	Unique + Not NULL	No
Unique	All values unique	Yes
Not NULL	No NULL values	No
Check	Values satisfy condition	Yes
Foreign Key	Referential integrity	Yes
Default	Default value set	Yes
Composite Key	Uniqueness of combination	No

OPERATIONS ON TABLE
WITH CONSTRAINTS

Constraints enforce rules on data. When we perform operations on a table, these rules are checked to maintain accuracy, consistency and integrity.

SAMPLE TABLE: STUDENT

```
CREATE TABLE Student (
  RollNo      INT PRIMARY KEY,
  Email       VARCHAR(100) UNIQUE,
  Name        VARCHAR(50) NOT NULL,
  Age         INT CHECK (Age >= 18),
  DeptID      INT
  FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);
```

CONSTRAINTS IN THIS TABLE

- **PRIMARY KEY** (RollNo) – Uniquely identifies each row. No NULL, no duplicate.
- **UNIQUE** (Email) – All emails must be unique.
- **NOT NULL** (Name) – Name cannot have NULL value.
- **CHECK** (Age >= 18) – Age must be 18 or greater.
- **FOREIGN KEY** (DeptID) – DeptID must exist in Department table.

DEPARTMENT TABLE (REFERENCE)

DeptID	DeptName
1	IT
2	HR
3	Sales

OPERATIONS AND EFFECT OF CONSTRAINTS

1 INSERT OPERATION

Adds new rows into the table.

EXAMPLE INSERT	RESULT	REASON
`INSERT INTO Student VALUES (101, 'asha@email.com', 'Asha', 20, 1);`	Success Row inserted.	All constraints satisfied.
`INSERT INTO Student VALUES (101, 'ravi@email.com', 'Ravi', 21, 2);`	Failed Duplicate RollNo.	Violates PRIMARY KEY (RollNo must be unique).
`INSERT INTO Student VALUES (102, 'asha@email.com', 'Asha', 22, 1);`	Failed Duplicate Email.	Violates UNIQUE constraint (Email must be unique).
`INSERT INTO Student VALUES (103, 'neha@email.com', NULL, 19, 1);`	Failed NULL in Name.	Violates NOT NULL (Name cannot be NULL).
`INSERT INTO Student VALUES (104, 'neha2@email.com', 'Neha', 17, 1);`	Failed Age less than 18.	Violates CHECK constraint (Age >= 18).
`INSERT INTO Student VALUES (105, 'karan@email.com', 'Karan', 20, 99);`	Failed Invalid DeptID.	Violates FOREIGN KEY (DeptID 99 does not exist).

2 UPDATE OPERATION

Modifies existing rows in the table.

EXAMPLE UPDATE	RESULT	REASON
`UPDATE Student SET Age = 22 WHERE RollNo = 101;`	Success Row updated.	Age remains >= 18.
`UPDATE Student SET Email = 'ravi@email.com' WHERE RollNo = 102;`	Failed Duplicate Email.	Violates UNIQUE constraint.
`UPDATE Student SET Name = NULL WHERE RollNo = 101;`	Failed NULL in Name.	Violates NOT NULL constraint.
`UPDATE Student SET Age = 15 WHERE RollNo = 103;`	Failed Age less than 18.	Violates CHECK constraint.
`UPDATE Student SET DeptID = 99 WHERE RollNo = 101;`	Failed Invalid DeptID.	Violates FOREIGN KEY constraint.

3 DELETE OPERATION

Removes rows from the table.

EXAMPLE DELETE	RESULT	REASON
`DELETE FROM Student WHERE RollNo = 101;`	Success Row deleted.	No constraint violation.
`DELETE FROM Department WHERE DeptID = 1;`	Failed Referential integrity error.	DeptID = 1 is used in Student table. (If ON DELETE RESTRICT)
`DELETE FROM Department WHERE DeptID = 1;`	Success Row deleted.	Allowed if foreign key has ON DELETE CASCADE.

HOW CONSTRAINTS AFFECT OPERATIONS

- Prevent invalid data from being inserted.
- Stop updates that break data rules.
- Restrict deletions that may break relationships.
- Maintain accuracy, consistency and integrity of the database.

COMMON CONSTRAINTS USED

- **PRIMARY KEY** – Uniquely identifies a row.
- **UNIQUE** – Ensures all values are unique.
- **NOT NULL** – Disallows NULL values.
- **CHECK** – Ensures values satisfy a condition.
- **FOREIGN KEY** – Ensures referential integrity.

KEY TAKEAWAY

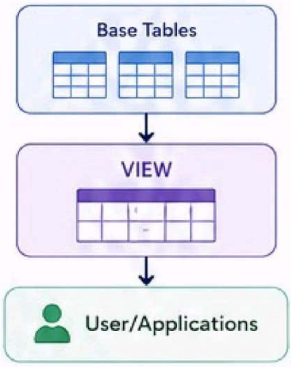
Every operation (INSERT, UPDATE, DELETE) is checked against the constraints of the table.
If a rule is broken, the operation is rejected to keep the data reliable and consistent.

Constraints + Operations = Reliable Data

VIEWS IN DBMS

NEED & ADVANTAGES

A VIEW is a virtual table based on the result of a SELECT query.
It does not store data physically.



WHAT IS A VIEW?

- A VIEW is a named, stored SELECT query.
- It presents data from one or more tables in a specific way.
- It provides a logical way to represent data.

EXAMPLE

Base Table: EMPLOYEE

EmpID	Name	DeptID	Salary
101	Asha	1	45000
102	Ravi	1	50000
103	Neha	2	48000
104	Karan	2	52000

(Shows only selected columns)

View: EMP_VIEW

EmpID	Name	Salary
101	Asha	45000
102	Ravi	50000
103	Neha	48000

```
CREATE VIEW EMP_VIEW AS SELECT EmpID, Name, Salary FROM EMPLOYEE;
```

NEED FOR VIEWS

1. SECURITY
Hide sensitive data by giving access only to required columns or rows.

2. SIMPLICITY
Simplify complex queries and present data in a user-friendly manner.

3. DATA ABSTRACTION
Provide a level of abstraction; users see what they need, not how data is stored.

4. DATA INDEPENDENCE
Changes in base tables do not affect the application if view structure remains same.

5. REUSABILITY
A view can be reused in multiple queries, reports and applications.

ADVANTAGES OF VIEWS

1. ENHANCED SECURITY
Restrict access to sensitive data. Users see only the data they are authorized to access.

2. SIMPLIFIED DATA ACCESS
Complex joins and calculations can be encapsulated in a view. Users can query the view instead of writing complex SQL.

3. DATA ABSTRACTION
Users are insulated from changes in the underlying tables. Only the view definition needs to be updated.

4. CONSISTENT DATA PRESENTATION
Views ensure consistent presentation of data across different users and applications.

5. REDUCED DATA REDUNDANCY
Data is stored in base tables only. Views do not store data, saving storage space.

6. IMPROVED PRODUCTIVITY
Developers can build views once and use them repeatedly, saving time and effort in writing queries.

EXAMPLE USE CASES

- HR department can view only employee details without salary.
- Managers can view department-wise summaries.
- Applications can use views to fetch required data easily.

UPDATABILITY OF VIEWS

Views can be updatable if:

- ✅ They are based on a single table
- ✅ They do not use DISTINCT, GROUP BY, aggregate functions, etc.

- ❌ Otherwise, they are read-only.

IMPORTANT NOTE

A View is a virtual table.
Any change in the base table data is reflected in the view automatically.

Views provide security, simplicity and abstraction, making database systems more powerful and user-friendly.

VIEWS IN DBMS

TYPES OF VIEWS

A view is a **virtual table** based on the **result of a SELECT query.**
Views can be of different types based on their function and characteristics.

BASE TABLES (Example)

STUDENT

RollNo	Name	DeptID	City
101	Asha	1	Pune
102	Ravi	1	Mumbai
103	Neha	2	Nagpur
104	Karan	2	Pune

DEPARTMENT

DeptID	DeptName	HOD
1	Computer Science	Dr. Mehta
2	Electronics	Dr. Rao

SALARY

RollNo	Salary
101	45000
102	50000
103	48000
104	52000

TYPES OF VIEWS

1 SIMPLE VIEW

A view based on a single table.
It may include some or all columns of the table.

Example SQL

```
CREATE VIEW VW_STUDENT AS
SELECT RollNo, Name, City
FROM STUDENT;
```

Result (View Output)

RollNo	Name	City
101	Asha	Pune
102	Ravi	Mumbai
103	Neha	Nagpur
104	Karan	Pune

2 COMPLEX VIEW

A view based on multiple tables using JOIN.
It combines data from related tables.

Example SQL

```
CREATE VIEW VW_STUDENT_DEPT AS
SELECT S.RollNo, S.Name, D.DeptName
FROM STUDENT S
JOIN DEPARTMENT D ON S.DeptID = D.DeptID;
```

Result (View Output)

RollNo	Name	DeptName
101	Asha	Computer Science
102	Ravi	Computer Science
103	Neha	Electronics
104	Karan	Electronics

3 DERIVED VIEW

A view based on expressions, calculations or derived columns.

Example SQL

```
CREATE VIEW VW_SALARY_BONUS AS
SELECT RollNo, Salary,
    Salary * 0.10 AS Bonus
FROM SALARY;
```

Result (View Output)

RollNo	Salary	Bonus
101	45000	4500
102	50000	5000
103	48000	4800
104	52000	5200

4 AGGREGATE VIEW

A view that uses aggregate functions like COUNT(), SUM(), AVG(), etc.

Example SQL

```
CREATE VIEW VW_DEPT_SUMMARY AS
SELECT D.DeptName, COUNT(S.RollNo) AS TotalStudents,
       AVG(SALARY) AS AvgSalary
FROM STUDENT S
JOIN DEPARTMENT D ON S.DeptID = D.DeptID
JOIN SALARY SA ON S.RollNo = SA.RollNo
GROUP BY D.DeptName;
```

Result (View Output)

DeptName	TotalStudents	AvgSalary
Computer Science	2	47500.00
Electronics	2	50000.00

5 INLINE VIEW

A view defined within another query.
(Also called subquery in FROM clause)

Example SQL

```
SELECT V.Name, V.Salary
FROM (SELECT S.Name, SA.Salary
      FROM STUDENT S
      JOIN SALARY SA ON S.RollNo = SA.RollNo)
      V
WHERE V.Salary > 48000;
```

Result (Output)

Name	Salary
Ravi	50000
Karan	52000

6 MATERIALIZED VIEW

A view whose result is physically stored.
It is a snapshot of data at a point of time.
(Supported in some DBMS)

Example SQL (Oracle Syntax)

```
CREATE MATERIALIZED VIEW MV_SALARY_SUM
BUILD IMMEDIATE
REFERSH COMPLETE ON DEMAND
AS
SELECT DeptID, SUM(Salary) AS TotalSalary
FROM SALARY S
JOIN STUDENT ST ON S.RollNo = ST.RollNo
GROUP BY DeptID;
```

Result (View Output)

DeptID	TotalSalary
1	95000
2	100000

SUMMARY COMPARISON

View Type	Based On	Uses	Updatable	Stores Data Physically	Example
Simple View	Single table	Hide columns, simplify data	Yes	No	VW_STUDENT
Complex View	Multiple tables (JOIN)	Combine related data	Sometimes	No	VW_STUDENT_DEPT
Derived View	Expressions / Calculations	Computed columns	Sometimes	No	VW_SALARY_BONUS
Aggregate View	Aggregate functions	Summary / Reports	No	No	VW_DEPT_SUMMARY
Inline View	Subquery in FROM	Temporary result set	–	No	(Used inside query)
Materialized View	Query result snapshot	Performance improvement	No	Yes	MV_SALARY_SUM

KEY TAKEAWAY

- Views do not store data (except materialized views).
- They provide data abstraction, security, and simplify complex queries.
- Different types of views serve different purposes in database systems.

Views make databases more secure, flexible and user-friendly by showing the right data in the right way.

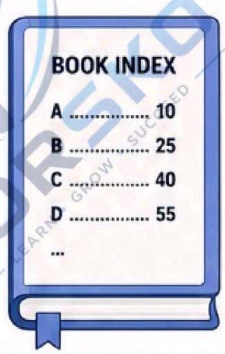
INDEX IN DBMS

NEED, TYPES & ADVANTAGES

An **Index** is a database object that improves the speed of data retrieval operations on a table, at the cost of additional storage space.

WHAT IS AN INDEX?

- An index is a data structure that improves the speed of searching and retrieving rows from a table.
- Indexes are like the index of a book, which helps find the desired topic without scanning every page.
- Indexes are automatically used by the DBMS when they can improve the performance of a query.



NEED FOR INDEX

Faster Data Retrieval
Indexes reduce the number of disk I/O operations by quickly locating the required rows.

Improved Query Performance
Speeds up SELECT queries, especially on large tables.

Efficient Sorting & Grouping
Helps in ORDER BY, GROUP BY and DISTINCT operations.

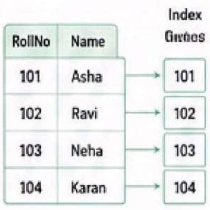
Faster Joins
Indexes on join columns improve the performance of JOIN operations.

Enforce Uniqueness
Unique indexes help in enforcing PRIMARY KEY and UNIQUE constraints.

TYPES OF INDEXES

1 PRIMARY INDEX

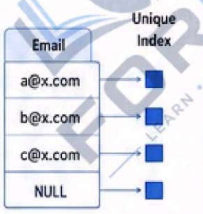
Index is created on the primary key of an ordered file (usually on clustered data). One primary index per table.



- Built on primary key
- Data is stored in sorted order

2 UNIQUE INDEX

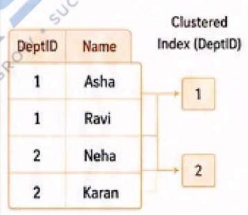
Ensures all values in the indexed column(s) are unique. Allows at most one NULL value.



- Enforces uniqueness
- Used for UNIQUE constraint

3 CLUSTERED INDEX

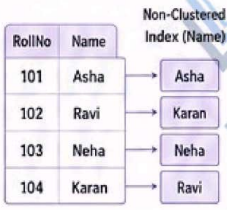
The data rows are stored in the order of the clustered index key. Only one clustered index per table.



- Data physically stored in index order

4 NON-CLUSTERED INDEX

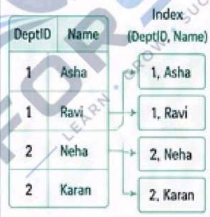
Index is created on a column that does not define the physical order of data.



- Data and index are stored separately

5 COMPOSITE INDEX

Index is created on two or more columns.



- Useful for multi-column search conditions

6 FULL-TEXT INDEX

Used to search text within large text data (varchar, char, text). Supported in some DBMS (MySQL, SQL Server).

- Optimized for text search
- Supports natural language search

ADVANTAGES OF INDEXES

Faster Data Retrieval
Greatly reduces the time taken to search and retrieve data.

Improved Query Performance
Speeds up complex queries, joins, sorting and filtering.

Efficient Sorting & Grouping
Helps in ORDER BY, GROUP BY and DISTINCT operations.

Better Scalability
Maintains good performance even as data grows.

Enforce Constraints
Helps enforce PRIMARY KEY and UNIQUE constraints efficiently.

Better User Experience
Applications become faster and more responsive.

IMPORTANT POINTS

- Indexes improve read performance but may slow down INSERT, UPDATE, DELETE operations.
- Indexes require extra storage space.
- Too many indexes can degrade overall performance.
- Always create indexes on columns used in WHERE, JOIN, ORDER BY, GROUP BY clauses.
- The DBMS automatically decides whether to use an index or not.

EXAMPLE

Create Non-Clustered Index

```
CREATE INDEX idx_name ON Student(Name);
```

Create Composite Index

```
CREATE INDEX idx_dept_name ON Student(DeptID, Name);
```

Indexes are powerful tools to boost database performance, but they should be used wisely and only when needed.

SQL JOINS

Combine rows from two or more tables based on a related column between them.

Example Tables

STUDENT (S)

ID	Name	DeptID
1	Asha	10
2	Ravi	20
3	Neha	30
4	Karan	NULL

DEPARTMENT (D)

DeptID	DeptName
10	Computer
20	Electronics
40	IT

SYNTAX

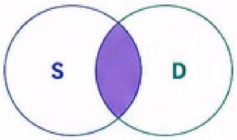
```
SELECT columns
FROM Table1 [JOIN TYPE] JOIN Table2
ON Table1.column = Table2.column;
```

Blue : Table 1 (STUDENT)
Green : Table 2 (DEPARTMENT)
Gray : No Match

TYPES OF JOINS

1 INNER JOIN

Returns only matching rows from both tables.



SQL

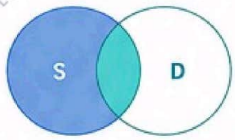
```
SELECT * FROM Student S
INNER JOIN Department D
ON S.DeptID = D.DeptID;
```

Result

ID	Name	DeptID	DeptName
1	Asha	10	Computer
2	Ravi	20	Electronics

2 LEFT JOIN

Returns all rows from the left table (S) and matching rows from the right table (D).



SQL

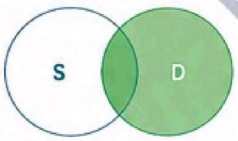
```
SELECT * FROM Student S
LEFT JOIN Department D
ON S.DeptID = D.DeptID;
```

Result

ID	Name	DeptID	DeptName
1	Asha	10	Computer
2	Ravi	20	Electronics
3	Neha	30	NULL
4	Karan	NULL	NULL

3 RIGHT JOIN

Returns all rows from the right table (D) and matching rows from the left table (S).



SQL

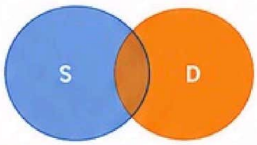
```
SELECT * FROM Student S
RIGHT JOIN Department D
ON S.DeptID = D.DeptID;
```

Result

ID	Name	DeptID	DeptName
1	Asha	10	Computer
2	Ravi	20	Electronics
NULL	NULL	40	IT

4 FULL OUTER JOIN

Returns all rows when there is a match in either left (S) or right (D) table. Unmatched rows are NULL.



SQL

```
SELECT * FROM Student S
FULL OUTER JOIN Department D
ON S.DeptID = D.DeptID;
```

Result

ID	Name	DeptID	DeptName
1	Asha	10	Computer
2	Ravi	20	Electronics
3	Neha	30	NULL
4	Karan	NULL	NULL
NULL	NULL	40	IT

5 CROSS JOIN

Returns the Cartesian product of both tables.

SQL

```
SELECT * FROM Student S
CROSS JOIN Department D;
```

Result (Total Rows = 4 × 3 = 12)

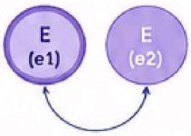
ID	Name	DeptID	DeptName
1	Asha	10	Computer
1	Asha	20	Electronics
1	Asha	40	IT
2	Ravi	10	Computer
...	...	...	...

6 SELF JOIN

Joins a table with itself. Useful for hierarchical data.

EMPLOYEE (E)

EmpID	Name	ManagerID
1	Alice	NULL
2	Bob	1
3	Carol	1
4	David	2



SQL

```
SELECT e1.Name AS Employee, e2.Name AS Manager
FROM Employee e1
LEFT JOIN Employee e2
 ON e1.ManagerID = e2.EmpID;
```

Result

Employee	Manager	NdLL
Alice	Nargar	Alice
Bob	Alice	Alice
David	Bolo	Bob

7 NATURAL JOIN

Automatically joins tables using all columns with the same name and data type.

TABLE A

ID	Name
1	Asha
2	Ravi
3	Neha

=

TABLE B

ID	City
1	Pune
2	Mumbai
4	Nagpur

SQL

```
SELECT * FROM TableA
NATURAL JOIN TableB;
```

Result

ID	Name	City
1	Asha	Pune
2	Ravi	Mumbai

JOIN TYPES COMPARISON

Join Type	Matches	Left Table Rows	Right Table Rows	Unmatched Rows Included?
INNER JOIN	Only matches	Yes	Yes	No
LEFT JOIN	All left + matches	Yes	Yes	Left only
RIGHT JOIN	All right + matches	Yes	Yes	Right only
FULL OUTER JOIN	All + matches	Yes	Yes	Both sides
CROSS JOIN	All combinations	Yes	Yes	No (all paired)
SELF JOIN	Within same table	Yes	Yes	Depends
NATURAL JOIN	Based on same column names	Yes	Yes	No

KEY TAKEAWAYS

- INNER JOIN returns only matching rows.
- LEFT JOIN returns all from left, matched from right.
- RIGHT JOIN returns all from right, matched from left.
- FULL OUTER JOIN returns all rows from both tables.
- CROSS JOIN returns all possible combinations.
- SELF JOIN is used to join a table with itself.
- NATURAL JOIN automatically matches same-named columns.

Joins help you combine data from multiple tables to get meaningful information.
Choose the right join type based on your requirement.

TYPES OF JOINS IN SQL

Joins are used to combine rows from two or more tables based on a related column between them.

Table: EMPLOYEE (E)

EmpID	Name	DeptID
1	Asha	10
2	Ravi	20
3	Neha	30
4	Karan	NULL

Table: DEPARTMENT (D)

DeptID	DeptName	Location
10	Computer	Pune
20	Electronics	Mumbai
30	IT	Nagpur
40	HR	Delhi

= Table E (Employee)
= Table D (Department)
= Matching Rows

1 INNER JOIN

Returns only the rows that have matching values in both tables.

Venn Diagram

SQL Syntax

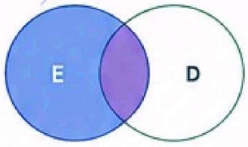
```
SELECT E.EmpID, E.Name, D.DeptName
FROM Employee E
INNER JOIN Department D
ON E.DeptID = D.DeptID;
```

Result (E JOIN D)

EmpID	Name	DeptName
1	Asha	Computer
2	Ravi	Electronics
3	Neha	IT

2 LEFT JOIN

Returns all rows from the left table (E) and the matching rows from the right table (D).



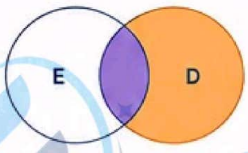

```
SELECT E.EmpID, E.Name, D.DeptName
FROM Employee E
LEFT JOIN Department D
ON E.DeptID = D.DeptID;
```

Result (E LEFT JOIN D)

EmpID	Name	DeptName
1	Asha	Computer
2	Ravi	Electronics
3	Neha	IT
4	Karan	NULL

3 RIGHT JOIN

Returns all rows from the right table (D) and the matching rows from the left table (E).




```
SELECT E.EmpID, E.Name, D.DeptName
FROM Employee E
RIGHT JOIN Department D
ON E.DeptID = D.DeptID;
```

Result (E RIGHT JOIN D)

EmpID	Name	DeptName
1	Asha	Computer
2	Ravi	Electronics
3	Neha	IT
NULL	NULL	HR

4 FULL OUTER JOIN

Returns all rows when there is a match in either table. Unmatched rows will contain NULL.

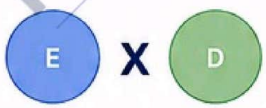

```
SELECT E.EmpID, E.Name, D.DeptName
FROM Employee E
FULL OUTER JOIN Department D
ON E.DeptID = D.DeptID;
```

Result (FULL OUTER JOIN)

EmpID	Name	DeptName
1	Asha	Computer
2	Ravi	Electronics
3	Neha	IT
4	Karan	NULL
NULL	NULL	HR

5 CROSS JOIN

Returns the Cartesian product of both tables. All combinations of rows.




```
SELECT E.EmpID, E.Name, D.DeptName
FROM Employee E
CROSS JOIN Department D;
```

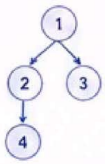
Result (CROSS JOIN)

EmpID	Name	DeptName
1	Asha	Computer
1	Asha	Electronics
1	Neha	IT
1	Asha	HR
...	...	...

6 SELF JOIN

Joins a table with itself. Useful for hierarchical data.

EmpID	Name	ManagerID
1	Alice	NULL
2	Bob	1
3	Carol	1
4	David	2




```
SELECT E1.Name AS Employee,
   E2.Name AS Manager
FROM Employee E1
LEFT JOIN Employee E2
ON E1.ManagerID = E2.EmpID;
```

Result (SELF JOIN)

Employee	Manager
Alice	NULL
Bob	Alice
Carol	Alice
David	Bob

7 NATURAL JOIN

Automatically joins tables using columns that have the same name and data type.

```
SELECT *
FROM Employee
NATURAL JOIN Department;
```

Result (NATURAL JOIN)

EmpID	Name	DeptID	DeptName
1	Asha	10	Computer
2	Ravi	20	Electronics
3	Neha	30	IT

QUICK COMPARISON

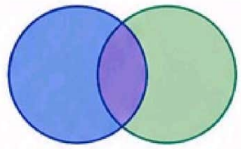
Join Type	Returns	Matched Rows	Unmatched from Left (E)	Unmatched from Right (D)
INNER JOIN	Only matching rows	Yes	No	No
LEFT JOIN	All from left + matched right	Yes	Yes	No
RIGHT JOIN	All from right + matched left	Yes	No	Yes
FULL OUTER JOIN	All from both sides	Yes	Yes	Yes
CROSS JOIN	All combinations	Yes	Yes	Yes
SELF JOIN	Table joined with itself	Yes	Depends	Depends
NATURAL JOIN	Auto join on same column names	Yes	No	No

KEY TAKEAWAYS

- INNER JOIN returns only matching rows.
- LEFT JOIN keeps all rows from the left table.
- RIGHT JOIN keeps all rows from the right table.
- FULL OUTER JOIN keeps everything from both tables.
- CROSS JOIN returns all possible combinations.
- SELF JOIN is used to compare rows within the same table.
- NATURAL JOIN matches same-named columns automatically.

Choose the right join type to get the data you need!

UNION OF TABLES AND ITS TYPES



The UNION operator is used to combine the result sets of two or more SELECT statements.

TABLE: EMPLOYEE

EmpID	Name	Department
1	Asha	IT
2	Ravi	IT
3	Neha	HR
4	Karan	HR

TABLE: CONTRACTOR

EmpID	Name	Department
3	Neha	HR
4	Karan	HR
5	Sohan	Admin
6	Pooja	Admin

BASIC UNION SYNTAX

```
SELECT column1, column2, ... FROM table1
UNION
SELECT column1, column2, ... FROM table2;
```

Both SELECT statements must have:
- Same number of columns
- Similar data types
- Columns in the same order

TYPES OF UNION

1 UNION (DISTINCT)

Combines the result sets and eliminates duplicate rows. (Default behavior)

Syntax

```
SELECT column_list FROM table1
UNION
SELECT column_list FROM table2;
```

Example: EMPLOYEE ∪ CONTRACTOR (Using UNION)

EMPLOYEE

EmpID	Name	Department
1	Asha	IT
2	Ravi	IT
3	Neha	HR
4	Karan	HR

CONTRACTOR

EmpID	Name	Department
3	Neha	HR
4	Karan	HR
5	Sohan	Admin
6	Pooja	Admin

UNION →

RESULT (UNION)

EmpID	Name	Department
1	Asha	IT
2	Ravi	IT
3	Neha	HR
4	Karan	HR
5	Sohan	Admin
6	Pooja	Admin

Duplicate rows (Neha, Karan) appear only once.

2 UNION ALL

Combines the result sets and includes all rows, including duplicates.

Syntax

```
SELECT column_list FROM table1
UNION ALL
SELECT column_list FROM table2;
```

Example: EMPLOYEE ∪ CONTRACTOR (Using UNION ALL)

EMPLOYEE

EmpID	Name	Department
1	Asha	IT
2	Ravi	IT
3	Neha	HR
4	Karan	HR

CONTRACTOR

EmpID	Name	Department
3	Neha	HR
4	Karan	HR
5	Sohan	Admin
6	Pooja	Admin

UNION ALL →

RESULT (UNION ALL)

EmpID	Name	Department
1	Asha	IT
2	Ravi	IT
3	Neha	HR
4	Karan	HR
3	Neha	HR
4	Karan	HR
5	Sohan	Admin
6	Pooja	Admin

Duplicate rows (Neha, Karan) appear as many times as they occur.

3 UNION CORRESPONDING BY (SQL Standard)

Combines result sets by matching columns with the same names, not just by position.

Syntax

```
SELECT column_list FROM table1
UNION CORRESPONDING BY (column_name, ...)
SELECT column_list FROM table2;
```

Example: UNION CORRESPONDING BY (Name_ Department)

TABLE A

EmpID	Name	Department
1	Asha	IT
2	Ravi	IT
3	Neha	HR

TABLE B

Name	EmpID	Department
Neha	3	HR
Karan	4	HR
Sohan	5	Admin

UNION CORRESPONDING BY (Name, Department) →

RESULT

Name	Department
Asha	IT
Ravi	IT
Neha	HR
Karan	HR
Sohan	Admin

Matching is done by column names (Name, Department), not by position.

IMPORTANT POINTS

- UNION removes duplicates.
- UNION ALL keeps all duplicates.
- Both SELECT statements must have the same number of columns and compatible data types.
- Column names in the result are taken from the first SELECT.
- ORDER BY can be used only at the end of the final result.

Example with ORDER BY

```
SELECT Name FROM Employee
UNION
SELECT Name FROM Contractor
ORDER BY Name;
```

QUICK COMPARISON

Type	Duplicates	Column Match	Performance	Use When
UNION	Removes duplicates	By position	Slower (due to duplicate check)	You want unique records only
UNION ALL	Keeps duplicates	By position	Faster	You want all records, including duplicates
UNION CORRESPONDING BY	Removes duplicates	By column name	Varies	Column order may be different in tables

Use UNION for unique results, UNION ALL for complete results including duplicates, and UNION CORRESPONDING BY when column order is different.

IN-BUILT FUNCTIONS

CHARACTER FUNCTIONS & NUMBER FUNCTIONS

In-built functions are predefined functions in DBMS that perform operations on values and return a result.

A CHARACTER FUNCTIONS (String Functions)

These functions are used to perform operations on character or string data.

Function	Description	Syntax	Example	Result
UPPER()	Converts a string to upper case.	UPPER(string)	UPPER('forsko')	'FORSKO'
LOWER()	Converts a string to lower case.	LOWER(string)	LOWER('FORSkO')	'forsko'
INITCAP()	Converts the first character of each word to upper case.	INITCAP(string)	INITCAP('hello world from dbms')	'Hello World From Dbms'
LENGTH() or LEN()	Returns the length (number of characters) of a string.	LENGTH(string) LEN(string)	LENGTH('forsko')	6
SUBSTR() or SUBSTRING()	Extracts a part of a string.	SUBSTR(string, start, length)	SUBSTR('database', 1, 4)	'data'
INSTR() or CHARINDEX()	Returns the position of a substring in a string.	INSTR(string, substring) (Starts from 1)	INSTR('database', 'base')	5
LTRIM()	Removes leading (spaces) from a string.	LTRIM(string)	LTRIM(' forsko')	'forsko'
RTRIM()	Removes trailing (spaces) from a string.	RTRIM(string)	RTRIM('forsko ')	'forsko'
TRIM()	Removes leading and trailing spaces.	TRIM(string)	TRIM(' forsko ')	'forsko'
CONCAT()	Concatenates two or more strings.	CONCAT(str1, str2, ...)	CONCAT('Data', ' Base')	'Data Base'
REPLACE()	Replaces a substring with another string.	REPLACE(string, old, new)	REPLACE('dbms is fun','fun', 'easy')	'dbms is easy'
REVERSE()	Reverses a string.	REVERSE(string)	REVERSE('hello')	'olleh'
ASCII()	Returns ASCII value of the first character.	ASCII(string)	ASCII('A')	65
CHAR()	Returns character for the given ASCII value.	CHAR(ascii_code)	CHAR(65)	'A'

EXAMPLE QUERY (CHARACTER FUNCTIONS)

```
SELECT Name,
       UPPER(Name) AS UpperName,
       LENGTH(Name) AS NameLength,
       SUBSTR(Name, 1, 3) AS First3Chars
FROM   Student;
```

Student Table

Name
Asha
Ravi
Neha

Output (Partial)

Name	UpperName	NameLength	First3Chars
Asha	ASHA	4	Ash
Ravi	RAVI	4	Rav
Neha	NEHA	4	Neh

B NUMBER FUNCTIONS

These functions are used to perform operations on numeric data.

Function	Description	Syntax	Example	Result
ABS()	Returns the absolute value.	ABS(number)	ABS(-25)	25
SQRT()	Returns the square root.	SQRT(number)	SQRT(16)	4
POWER() or POW()	Returns x raised to the power y.	POWER(x, y) POW(x, y)	POWER(2, 3)	8
ROUND()	Rounds a number to a specified number of decimal places.	ROUND(number, decimals)	ROUND(45.678,2)	45.68
CEIL() or CEILING()	Returns the smallest integer greater than or equal to the number.	CEIL(number) CEILING(number)	CEIL(7.2)	8
FLOOR()	Returns the largest integer less than or equal to the number.	FLOOR(number)	FLOOR(7.8)	7
TRUNC()	Truncates the decimal part of a number.	TRUNC(number, decimals)	TRUNC(45.678)	45
MOD() or (Modulus)	Returns the remainder after division.	MOD(a, b) a % b	MOD(10,3) 10 % 3	1
SIGN()	Returns the sign of a number.	SIGN(number)	SIGN(-10)	-1
EXP()	Returns e raised to the given power.	EXP(number)	EXP(1)	2.71828
LOG() or LN()	Returns logarithm (base 10) or natural logarithm.	LOG(number) LN(number)	LOG(100) LN(2.71828)	2 1
GREATEST()	Returns the largest value in the list.	GREATEST(a, b, c, ...)	GREATEST(5,7,3)	7
LEAST()	Returns the smallest value in the list.	LEAST(a, b, c, ...)	LEAST(5,7,3)	3

EXAMPLE QUERY (NUMBER FUNCTIONS)

```
SELECT Marks,
       ABS(Marks) AS AbsMarks,
       ROUND(Marks,1) AS Rounded,
       SQRT(Marks) AS SquareRoot,
       MOD(Marks,3) AS ModResult
FROM   Exam;
```

Exam Table

Marks
25
-16
49.5

Output (Partial)

Marks	AbsMarks	Rounded	SquareRoot	ModResult
25	25	25.0	5	1
-16	-16	16	-16.0	4	2
49.5	49.5	49.5	49.5	7.03	1

KEY POINTS

- Character functions work on string (CHAR, VARCHAR, TEXT) data.
- Number functions work on numeric (INT, FLOAT, DECIMAL, etc.) data.
- These functions make queries powerful and results more meaningful.
- They are widely used in WHERE, SELECT, ORDER BY, GROUP BY, HAVING clauses.

COMMONLY USED TOGETHER

```
ROUND(AVG(Marks),2)
UPPER(TRIM(Name))
LENGTH(CONCAT(FirstName, ' ', LastName))
MOD(TotalMarks,2) = 0  → Check Even/Odd
```

In-built functions save time, reduce complexity and make your SQL queries smarter!

IN-BUILT FUNCTIONS IN DBMS

DATE FUNCTIONS & GROUP FUNCTIONS

In-built functions are predefined functions in DBMS that perform operations on values and return a result.

A DATE FUNCTIONS

These functions are used to perform operations on date and time data.

Function	Description	Syntax	Example	Result
CURRENT_DATE()	Returns the current date.	CURRENT_DATE()	SELECT CURRENT_DATE();	2025-05-15
CURRENT_TIME()	Returns the current time.	CURRENT_TIME()	SELECT CURRENT_TIME();	14:30:25
CURRENT_TIMESTAMP()	Returns the current date and time.	CURRENT_TIMESTAMP()	SELECT CURRENT_TIMESTAMP();	2025-05-15 14:30:25
DATE()	Extracts date part from a datetime value.	DATE(datetime)	SELECT DATE('2025-05-15 14:30:25');	2025-05-15
TIME()	Extracts time part from a datetime value.	TIME(datetime)	SELECT TIME('2025-05-15 14:30:25');	14:30:25
YEAR()	Returns the year part.	YEAR(date)	SELECT YEAR('2025-05-15');	2025
MONTH()	Returns the month part (1-12).	MONTH(date)	SELECT MONTH('2025-05-15');	5
DAY()	Returns the day of month (1-31).	DAY(date)	SELECT DAY('2025-05-15');	15
HOUR()	Returns the hour part (0-23).	HOUR(time)	SELECT HOUR('14:30:25');	14
MINUTE()	Returns the minute part (0-59).	MINUTE(time)	SELECT MINUTE('14:30:25');	30
SECOND()	Returns the second part (0-59).	SECOND(time)	SELECT SECOND('14:30:25');	25
DATE_ADD()	Adds interval to a date.	DATE_ADD(date, INTERVAL expr unit)	SELECT DATE_ADD('2025-05-15', INTERVAL 10 DAY);	2025-05-25
DATE_SUB()	Subtracts interval from a date.	DATE_SUB(date, INTERVAL expr unit)	SELECT DATE_SUB('2025-05-15', INTERVAL 5 DAY);	2025-05-10
DATEDIFF()	Returns the number of days between two dates.	DATEDIFF(date1, date2)	SELECT DATEDIFF('2025-05-15', '2025-05-01');	14

EXAMPLE (DATE FUNCTIONS)

```
SELECT  Name,
        DOB,
        YEAR(DOB) AS BirthYear,
        AGE(CURDATE(), DOB) AS Age,
        DATE_ADD(DOB, INTERVAL 1 YEAR) AS NextBirthday
FROM    Student;
```

Output (Partial)

Name	DOB	BirthYear	Age	NextBirthday
Asha	2003-01-10	2003	22	2026-01-10
Ravi	2004-06-20	2004	20	2025-06-20
Neha	2003-11-05	2003	21	2026-11-05

DATE UNITS

YEAR	MONTH	HOUR	MINUTE	SECOND
INTERVAL 1 YEAR	INTERVAL 1 MONTH	INTERVAL 1 DAY	INTERVAL 1 MIOUR	INTERVAL 1 SECOND

B GROUP FUNCTIONS (Aggregate Functions)

These functions perform calculations on a set of values and return a single result.

Function	Description	Syntax	Example	Result
COUNT()	Returns the number of rows.	COUNT(*) or COUNT(column)	SELECT COUNT(*) FROM Student;	120
SUM()	Returns the sum of numeric values.	SUM(column)	SELECT SUM(Marks) FROM Exam;	6150
AVG()	Returns the average of numeric values.	AVG(column)	SELECT AVG(Marks) FROM Exam;	51.25
MIN()	Returns the minimum value.	MIN(column)	SELECT MIN(Marks) FROM Exam;	12
MAX()	Returns the maximum value.	MAX(column)	SELECT MAX(Marks) FROM Exam;	98
SUM(DISTINCT)	Returns sum of distinct values.	SUM(DISTINCT column)	SELECT SUM(DISTINCT Salary) FROM Emp;	1250000
COUNT(DISTINCT)	Returns the count of distinct values.	COUNT(DISTINCT column)	SELECT COUNT(DISTINCT DeptID) FROM Emp;	5

EXAMPLE (GROUP FUNCTIONS)

```
SELECT  DeptID,
        COUNT(*) AS TotalEmp,
        AVG(Salary) AS AvgSalary,
        MIN(Salary) AS MinSalary,
        MAX(Salary) AS MaxSalary,
        SUM(Salary) AS TotalSalary
FROM    Employee
GROUP BY DeptID;
```

Output

DeptID	TotalEmp	AvgSalary	MinSalary	MaxSalary	TotalSalary
10	25	45560.00	22000	90000	1139000
20	30	48266.67	21000	95000	1448000
30	20	39850.00	20000	85000	797000
40	15	51666.67	25000	100000	775000
50	30	46200.00	23000	90000	1386000

GROUP BY with HAVING

```
SELECT  DeptID, AVG(Salary) AS AvgSal
FROM    Employee
GROUP BY DeptID
HAVING  AVG(Salary) > 45000;
```

Output

DeptID	AvgSal
20	48266.67
40	51666.67
50	46200.00

KEY POINTS

- Group functions work on a set of values and return a single result.
- They are commonly used with GROUP BY to group rows.
- COUNT(*) counts all rows including NULLs.
- COUNT(column) counts only non-NULL values in that column.
- GROUP BY comes before HAVING in SQL execution.
- Use HAVING to filter groups (not rows).

QUICK COMPARISON

Aspect	Date Functions	Group Functions
Purpose	Work with date and time values	Work with groups of rows
Input	A date/time value	A set of values (column)
Output	A single date/time or part of it	A single aggregated value
Example	YEAR(DOB), DATE_ADD(DOB, INTERVAL 1 DAY)	COUNT(*), SUM(Salary), AVG(Marks)

COMMONLY USED TOGETHER

```
SELECT  DeptID,
        COUNT(*) AS TotalEmp,
        SUM(Salary) AS TotalSalary
FROM    Employee
WHERE   JoinDate >= '2025-01-01'
GROUP BY DeptID
HAVING  COUNT(*) > 5;
```

Use the right function for the right task to make your SQL queries efficient and powerful!

CONVERSION FUNCTIONS

IN SQL (DBMS)

Conversion functions are used to convert a value from one data type to another.

A → B
Convert Data Type

WHY USE CONVERSION FUNCTIONS?

- Ensure compatibility between different data types.
- Perform calculations on different data types.
- Display data in the required format.
- Avoid errors in implicit type conversion.

SYNTAX

```
FUNCTION ( expression AS target_data_type )
or
FUNCTION ( expression, target_data_type )
```

SUPPORTED DATA TYPES

- CHAR / VARCHAR / TEXT
- INTEGER / BIGINT / SMALLINT
- DECIMAL / NUMERIC / FLOAT
- DATE / TIME / TIMESTAMP

COMMON CONVERSION FUNCTIONS

Function	Description	Syntax	Example	Input (Type)	Output (Type)	Result
CAST()	Converts a value of one data type to another.	`CAST( expression AS data_type )`	`SELECT CAST('123' AS INT);`	'123' (TEXT)	INT	123
CONVERT()	Converts a value of one data type to another (using style for date).	`CONVERT( data_type, expression [, style])`	`SELECT CONVERT(INT, '456');`	'456' (TEXT)	INT	456
TO_CHAR()	Converts a value to a character string.	`TO_CHAR( expression [, format] )`	`SELECT TO_CHAR(12345.678, '99999.99');`	12345.678 (NUMBER)	VARCHAR	12345.68
TO_NUMBER()	Converts a value to a numeric value.	`TO_NUMBER( expression, [ format] )`	`SELECT TO_NUMBER('789.56');`	'789.56' (TEXT)	NUMBER	789.56
TO_DATE()	Converts a value to a DATE.	`TO_DATE( expression, 'format' )`	`SELECT TO_DATE('15-05-2025', 'DD-MM-YYYY');`	'15-05-2025' (TEXT)	DATE	15-MAY-25
TO_TIMESTAMP()	Converts a value to a TIMESTAMP.	`TO_TIMESTAMP( expression, 'format' )`	`SELECT TO_TIMESTAMP('15-05-2025 14:30:00', 'DD-MM-YYYY HH24:MI:SS');`	'15-05-2025 14:30:00' (TEXT)	TIMESTAMP	15-MAY-2025 14:30:00
TO_DATE (with style)	Converts a string to date using style (SQL Server).	`CONVERT(DATE, expression, style)`	`SELECT CONVERT(DATE, '05/15/2025', 101);`	'05/15/2025' (TEXT)	DATE	2025-05-15
TO_BINARY()	Converts a value to binary.	`CAST(expression AS BINARY(n) )`	`SELECT CAST('AB' AS BINARY(2));`	'AB' (TEXT)	BINARY	0x4142
TO_BOOLEAN()	Converts a value to Boolean (TRUE/FALSE).	`CAST(expression AS BOOLEAN )`	`SELECT CAST(1 AS BOOLEAN);`	1 (INT)	BOOLEAN	TRUE

EXAMPLE QUERIES

Purpose	Query	Output (Example)	Explanation
String to Integer	`SELECT CAST('100' AS INT);`	100	Converts string '100' to integer.
String to Date	`SELECT TO_DATE('2025-05-15','YYYY-MM-DD');`	15-MAY-25	Converts string to DATE.
Date to String	`SELECT TO_CHAR(SYSDATE,'DD-MON-YYYY');`	15-MAY-2025	Converts date to formatted string.
String to Number (Decimal)	`SELECT TO_NUMBER('123.45');`	123.45	Converts string to decimal number.
Number to String	`SELECT TO_CHAR(98765,'99999');`	98765	Converts number to string.
Date using CONVERT (SQL Server)	`SELECT CONVERT(DATE,'15/05/2025',103);`	2025-05-15	Converts string to date using style code.
Timestamp from String	`SELECT TO_TIMESTAMP('15-05-2025 14:30:00', 'DD-MM-YYYY HH24:MI:SS');`	15-MAY-2025 14:30:00	Converts string to timestamp.
Boolean Conversion	`SELECT CAST(0 AS BOOLEAN);`	FALSE	Converts number to Boolean.

KEY POINTS

- Conversion functions help in explicit conversion of data types.
- Always use correct format while converting dates.
- Invalid conversions may cause errors.
- CAST() is ANSI standard, CONVERT() is more flexible (SQL Server).
- Use TO_CHAR() and TO_NUMBER() mainly in Oracle.

COMMONLY USED TOGETHER

`TO_CHAR(SYSDATE, 'DD-MON-YYYY')`	→	Display date as string
`TO_DATE('15-05-2025', 'DD-MM-YYYY')`	→	Convert string to date
`CAST('100' AS INT)`	→	Convert string to integer
`TO_NUMBER('123.45')`	→	Convert string to number

Use conversion functions to ensure accurate calculations, proper comparisons, and formatted outputs in your queries!

USE OF ALIAS, NULL, NVL & SUB QUERIES

Simplify Queries • Handle Missing Data • Retrieve Smart Results

1 USE OF ALIAS

An alias is a temporary name given to a table or a column in a query. It is used to make queries easier to read and understand.

TYPES OF ALIAS

- **Column Alias** – Gives a new name to a column.
- **Table Alias** – Gives a short name to a table.

SYNTAX

Column Alias

```
SELECT column_name AS alias_name
FROM table_name;
```

Table Alias

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

EXAMPLE

Column Alias

```
SELECT emp_name AS "Employee Name",
       salary AS "Monthly Salary"
FROM employee;
```

Employee Name	Monthly Salary
Asha	45000
Ravi	52000
Neha	47000

Table Alias

```
SELECT e.emp_id, e.emp_name, e.salary
FROM employee e;
```

Alias makes queries shorter and improves readability.

2 NULL

NULL represents a missing, unknown or inapplicable value. It is not equal to zero or an empty string.

KEY POINTS

- NULL means no value.
- NULL cannot be compared using =.
- Use IS NULL or IS NOT NULL to check NULL.
- NULL in arithmetic operations returns NULL.

EXAMPLE TABLE

Student

RollNo	Name	Dept	Marks
1	Asha	IT	85
2	Ravi	CS	NULL
3	Neha	IT	78
4	Karan	CS	NULL

EXAMPLES

```
-- Students with NULL in Marks
SELECT * FROM Student WHERE Marks IS NULL;

-- Students with NOT NULL in Marks
SELECT * FROM Student WHERE Marks IS NOT NULL;
```

NOTES

- NULL is treated as unknown.
- COUNT(column_name) ignores NULL values.
- COUNT(*) counts all rows.

NULL COMPARISON

```
-- Wrong
SELECT * FROM Student WHERE Marks = NULL;

-- Correct
SELECT * FROM Student WHERE Marks IS NULL;
```

3 NVL (NULL VALUE)

NVL() is a function used to replace NULL with a specified value.

SYNTAX

```
NVL(expression, replacement_value)
```

- expression – The column or value to check.
- replacement_value – The value returned if expression is NULL.

EXAMPLE TABLE

Employee

EmpID	Name	Dept	Bonus
1	Asha	IT	5000
2	Ravi	CS	NULL
3	Neha	IT	NULL
4	Karan	CS	3000

EXAMPLES

```
-- Replace NULL bonus with 0
SELECT emp_id, name, NVL(bonus, 0) AS bonus
FROM employee;
```

EmpID	Name	Bonus
1	Asha	5000
2	Ravi	0
3	Neha	0
4	Karan	3000

```
-- Replace NULL dept with 'Not Assigned'
SELECT emp_id, name, NVL(dept, 'Not Assigned') AS dept
FROM employee;
```

EmpID	Name	Dept
1	Asha	IT
2	Ravi	CS
3	Neha	IT
4	Karan	CS

KEY POINTS

- NVL() returns the expression value if it is not NULL.
- If expression is NULL, it returns replacement_value.
- Helps in calculations and display with default values.

COMMON USES

Replace NULL salary with 0

```
SELECT NVL(salary, 0) FROM employee;
```

Replace NULL text with 'N/A'

```
SELECT NVL(address, 'N/A') FROM employee;
```

Replace NULL date with today's date

```
SELECT NVL(join_date, SYSDATE) FROM employee;
```

`NVL(expr, value)` is same as `IF expr IS NULL THEN value ELSE expr END`

4 SUB QUERIES

A sub query (or inner query) is a query nested inside another query. The inner query is executed first, and its result is used by the outer query.

TYPES OF SUB QUERIES

- **Single Row Sub Query** – Returns one row.
- **Multiple Row Sub Query** – Returns multiple rows.
- **Correlated Sub Query** – Refers to outer query.

SYNTAX

Single Row Sub Query

```
SELECT column(s)
FROM table
WHERE column = (sub query);
```

Multiple Row Sub Query

```
SELECT column(s)
FROM table
WHERE column IN (sub query);
```

Correlated Sub Query

```
SELECT column(s)
FROM table t1
WHERE EXISTS
   (SELECT 1 FROM table t2
    WHERE t2.col = t1.col);
```

EXAMPLES

1. Single Row Sub Query

```
-- Employees earning more than average salary
SELECT * FROM employee
WHERE salary > (SELECT AVG(salary) FROM employee);
```

2. Multiple Row Sub Query

```
-- Employees from departments located in location 'Pune' or 'Mumbai'
SELECT * FROM employee
WHERE dept_id IN (SELECT dept_id FROM department
                  WHERE location IN ('Pune', 'Mumbai'));
```

3. Correlated Sub Query

```
-- Employees who earn more than the average salary of their department
SELECT e1.* FROM employee e1
WHERE salary >  (SELECT AVG(salary)
                 FROM employee e2
                 WHERE e2.dept_id = e1.dept_id);
```

KEY POINTS

- Sub queries can be used in SELECT, FROM, WHERE, HAVING clauses.
- Use IN/ANY/ALL for multiple row sub queries.
- Use EXISTS for correlated sub queries.

Inner Query (Executed First) → Result → Outer Query (Uses the Result)

BENEFITS

- Break complex problems into smaller parts.
- Improve readability and maintainability.
- Powerful filtering and comparison.

Use these features to write smarter, cleaner and more efficient SQL queries!