

DBMS

MySQL

Dineshkumar Thangavel

Introduction: Why Database?

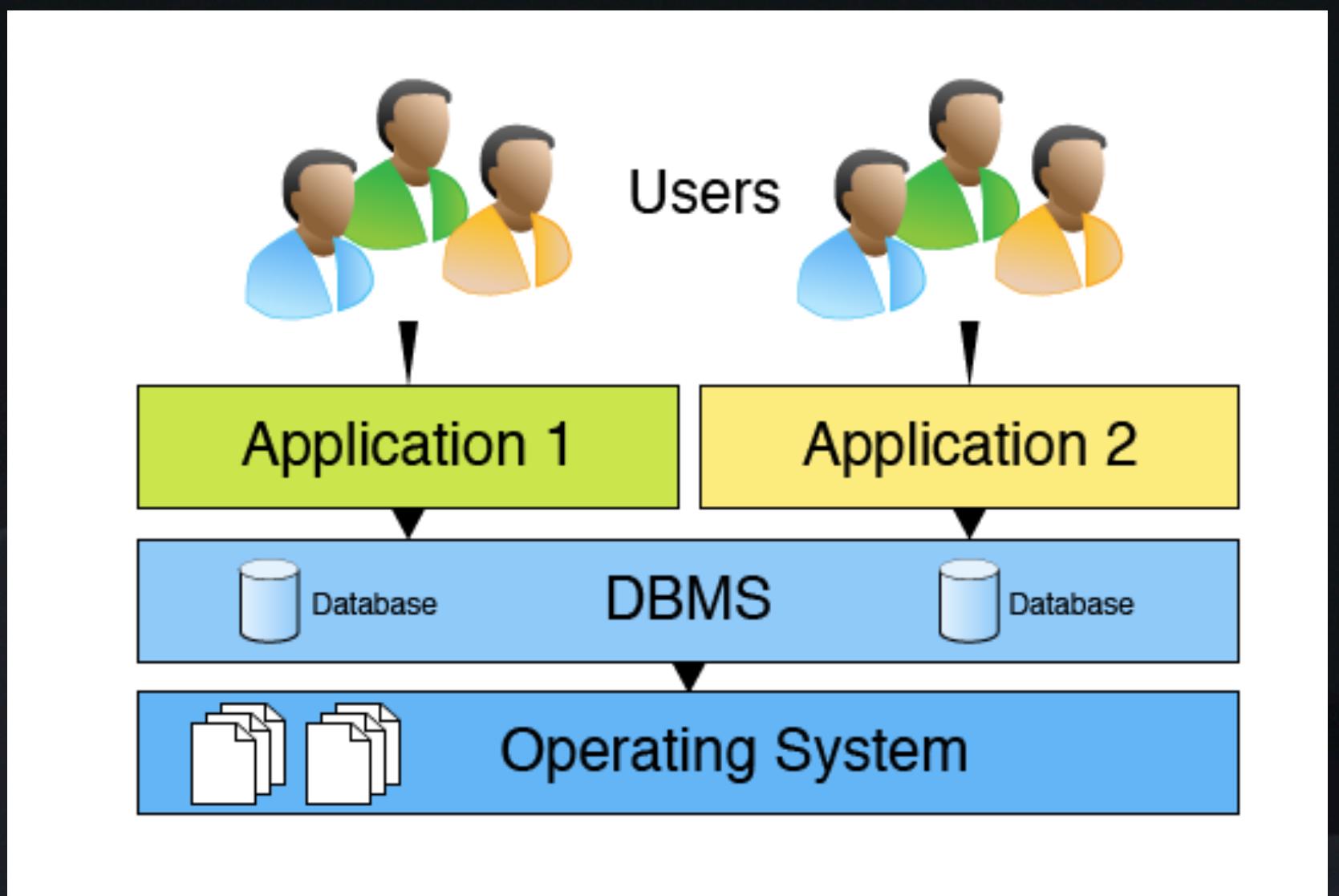
- Imagine your college stores student marks in Excel sheets – easy for 10 students, chaos for 10,000. Files get duplicated, lost, corrupted, and updating is a nightmare.
- **Database Management System (DBMS)** – a software to **store, organize, and manage** data efficiently.

Student_ID	Name	Dept	Marks
S101	Arun	CSE	89
S102	Meena	IT	95

```
SELECT Name, Marks FROM Student WHERE Dept = 'IT';
```

Database System Overview

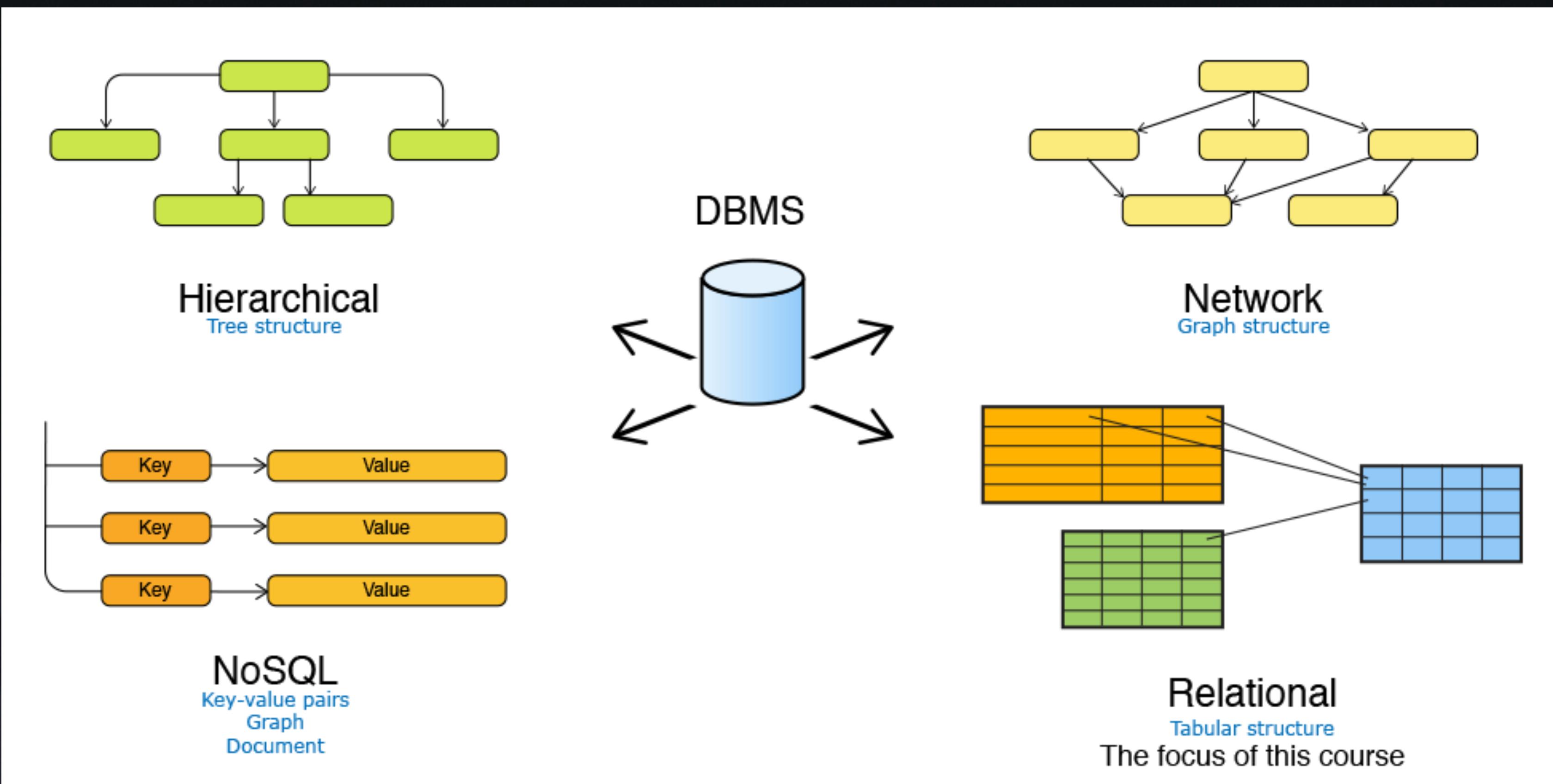
- A Database is a shared collection of logically related data and description of these data, designed to meet the information needs of an organization.
- A Database Management System(DBMS) is a software system that enables users to define, create, maintain, and control access to the database. Database Systems typically have high costs and require high-end hardware configurations.
- An Application Program communicates with a database by issuing an appropriate request (typically a SQL statement)



File System vs DBMS

Feature	File System	DBMS
Data Storage	In files (e.g., .txt, .csv)	In tables (rows & columns)
Redundancy	High	Reduced
Data Security	Poor	High (controlled access)
Data Sharing	Difficult	Multi-user support
Backup & Recovery	Manual	Automatic (Transaction logs)

Types of Database



DBMS vs RDBMS

Concept	DBMS	RDBMS
Data Storage	File-based	Table-based (relations)
Relationships	Not supported	Supported (via keys)
Example	dBase, FoxPro	MySQL, Oracle, PostgreSQL
Normalization	No	Yes
Query Language	Proprietary	SQL

DBMS Architecture

1. External Level (User View)

- What the user sees (e.g., marks report of their own department).

2. Conceptual Level (Logical View)

- Logical structure of the entire database (tables, relations).

3. Internal Level (Physical View)

- How data is stored physically (indexes, blocks, etc.)

Real-Life Analogy:

Like a college –

- Student sees marks (external view),
- Staff knows how subjects relate (conceptual),
- IT admin manages files on the server (internal).

Important DBMS Terms

Term	Description
Database	Organized collection of data
Table	Collection of rows & columns
Tuple (Row)	Single record
Attribute (Column)	Field of data
Schema	Structure definition
Instance	Actual data at a point in time

Keys in RDBMS

```
CREATE TABLE Department (  
    Dept_ID INT PRIMARY KEY,  
    Dept_Name VARCHAR(30)  
);
```

```
CREATE TABLE Student (  
    Student_ID INT PRIMARY KEY,  
    Name VARCHAR(30),  
    Dept_ID INT,  
    FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID)  
);
```

Key Type	Description	Example
Primary Key	Unique identifier	Student_ID
Foreign Key	Links to another table	Dept_ID in Student table
Candidate Key	All possible unique keys	Roll_No, Aadhaar_No
Super Key	Candidate + extra attributes	(Roll_No, Name)
Composite Key	Combination of columns	(Student_ID, Course_ID)

Schema vs Instance

- **Schema** → Structure (like class in OOP)
- **Instance** → Actual data (like objects)

Example:

```
CREATE TABLE Student (ID INT, Name VARCHAR(20));
```

Schema = column design

Instance = the actual rows inside it.

.

MySql Workbench Setup

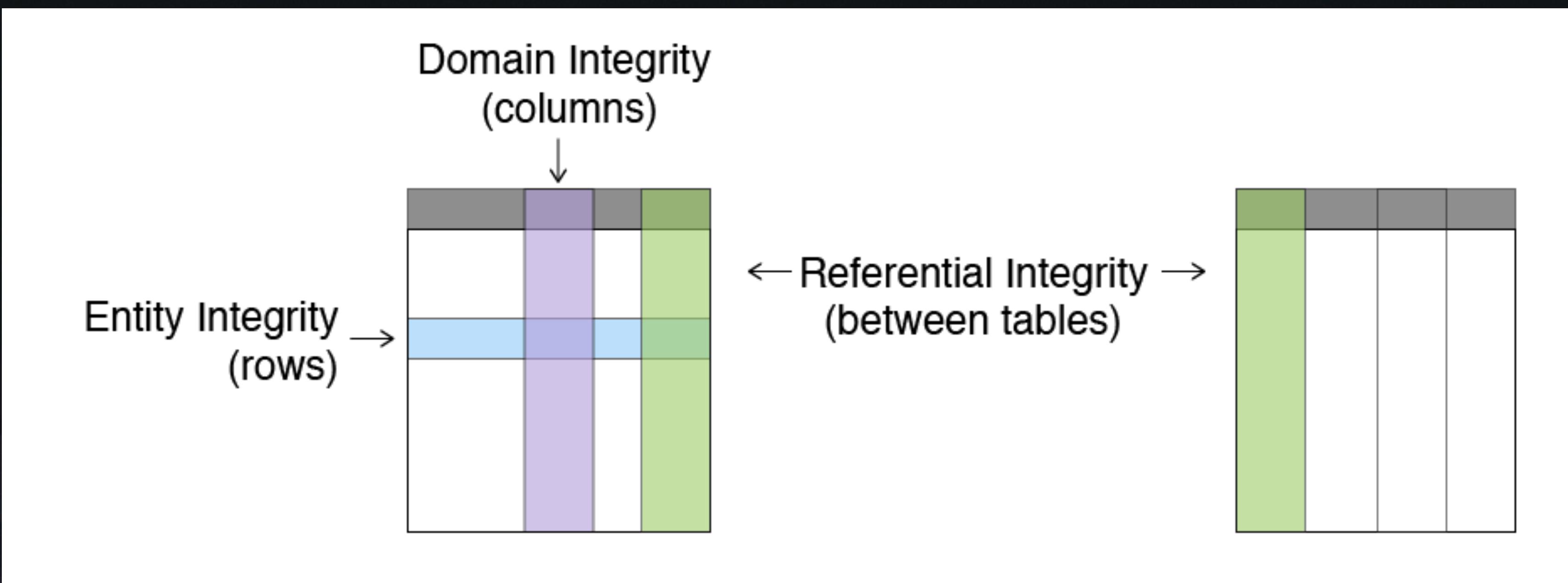
Steps:

1. Open MySQL Workbench → New Connection
2. Test Connection (root / password)
3. Create a new database
4. Create a table
5. Insert sample data
6. Retrieve data

Integrity Constraints in DBMS

Integrity Type	Definition	Example / Enforced Through
Entity Integrity	Ensures each row (record) in a table can be uniquely identified. A table must have a column or combination of columns (Primary Key) that cannot be NULL.	<ul style="list-style-type: none">◆ Enforced using PRIMARY KEY◆ Example: Student_ID in Student table uniquely identifies each student.
Domain Integrity	Ensures that the values in each column are valid and fall within a defined domain (data type, format, or business rule).	<ul style="list-style-type: none">◆ Enforced using DATA TYPE, CHECK, DEFAULT, and NOT NULL constraints.◆ Example: Gender CHAR(1) CHECK (Gender IN ('M','F'))
Referential Integrity	Ensures relationships between tables remain consistent — a foreign key value in one table must exist as a primary key value in the referenced table.	<ul style="list-style-type: none">◆ Enforced using FOREIGN KEY◆ Example: Dept_ID in Student must exist in Department(Dept_ID)

Integrity Constraints in DBMS



Why is an ER Model?

Definition:

- The Entity–Relationship (ER) model is a conceptual blueprint of your database – it shows how data entities relate to one another.

Simple Analogy:

- ER model is like the architect’s blueprint before constructing a building (database).
Tables = Rooms, Relationships = Doors connecting rooms.

Key Components of an ER Model

Component	Description	Example
Entity	A real-world object that we store data about.	Student, Course, LibraryBook
Attribute	Property or characteristic of an entity.	Name, Age, Dept
Relationship	Association between entities.	Student enrolls in Course

Types of Attributes

Attribute Type	Description	Example
Simple	Cannot be divided further.	Name, Age
Composite	Can be divided into subparts.	Address = (Street, City, Pincode)
Derived	Computed from other attributes.	Age derived from DOB
Multivalued	Can have multiple values.	Phone_Number

Types of Entities

Entity Type	Description	Example
Strong Entity	Exists independently (has its own primary key).	Student(Student_ID)
Weak Entity	Depends on another entity (no complete key).	Dependent of Employee

Types of Relationships

Notation:

- Diamond → Relationship
- Lines → Cardinality (1 or M)

Type	Meaning	Example	Cardinality
One-to-One (1:1)	Each entity of A is related to exactly one of B.	Each Student has one ID Card.	1:1
One-to-Many (1:N)	One record in A can relate to many in B.	Department → Students.	1:N
Many-to-Many (M:N)	Many in A can relate to many in B.	Student ↔ Course.	M:N

Cardinality in ER Model

Symbol	Meaning	Explanation	Example
	Exactly One (Mandatory One)	Each record must be associated with one and only one related record.	Every <i>Student</i> must belong to exactly one Department .
○	Zero or One (Optional One)	Relationship is optional — an entity may or may not be related, but if it is, only to one.	Each <i>Employee</i> may or may not have a <i>Parking Space</i> .
○<	Zero, One, or More (Optional Many)	An entity can be associated with none, one, or many records.	A <i>Customer</i> may or may not have <i>Orders</i> , but can have many.
<	One or More (Mandatory Many)	An entity must have at least one related record, and possibly many.	Each <i>Department</i> must have one or more <i>Students</i> .

Why Normalization?

Definition:

- Normalization is a process of organizing data in a database to reduce redundancy and improve data integrity.

Simple Analogy:

- Imagine a messy timetable pasted in 10 places in a college.
If one class timing changes, you have to update all 10 – that's redundancy and leads to inconsistency.
- Normalization fixes that – one source of truth.

Problems Before Normalization

Problem Type	Example	Description
Data Redundancy	Doctor name repeated for every patient	Same info stored multiple times
Update Anomaly	Change doctor's phone in one row, miss others	Data inconsistency
Insertion Anomaly	Can't add new doctor until a patient is added	Missing dependency
Deletion Anomaly	Delete last patient → lose doctor info	Unintended data loss

Term	Description	Example (Real-life Context)
Functional Dependency ($A \rightarrow B$)	Attribute A determines attribute B . Knowing A gives exactly one value of B.	Student_ID \rightarrow Name Each student ID uniquely determines a student name.
Determinant	Attribute (or set of attributes) that determines the value of another.	In Student_ID \rightarrow Name, Student_ID is the determinant.
Dependent	Attribute whose value depends on another attribute.	In Student_ID \rightarrow Name, Name is dependent on Student_ID.
Fully Functional Dependency	A non-key attribute depends on the <i>entire composite key</i> and not part of it.	$(\text{OrderID}, \text{ProductID}) \rightarrow \text{Quantity}$ Quantity depends on both OrderID & ProductID.
Partial Dependency	A non-key attribute depends on <i>part of a composite key</i> instead of the full key.	$(\text{OrderID}, \text{ProductID}) \rightarrow \text{Quantity}$ and $\text{ProductID} \rightarrow \text{ProductName} \Rightarrow \text{ProductName}$ partially depends on key.
Transitive Dependency	A non-key attribute depends on <i>another non-key attribute</i> .	Student_ID \rightarrow Dept_Name and Dept_Name \rightarrow Dept_Loc \Rightarrow Student_ID \rightarrow Dept_Loc (transitive).

Hospital Database (Unnormalized Form)

Observation:

- Doctor details are repeated for every patient.
- If Dr. Kiran changes phone number → multiple updates needed.

Patient_ID	Patient_Name	Doctor_Name	Doctor_Phone	Treatment	Room_No
P101	Arun	Dr. Kiran	9876543210	Fever	R101
P102	Meena	Dr. Kiran	9876543210	Cold	R102
P103	Kavi	Dr. Ravi	9786542109	Fracture	R103

Step 1: First Normal Form (1NF)

Rule:

- Each cell should hold atomic (single) values.
- No repeating groups or arrays.
- The Hospital table already satisfies atomicity since all values are single.

Patient_ID	Treatment	Room_No
P101	Fever, Cold	R101

However, for clarity:

- 1NF = Eliminate repeating groups → one value per cell.

💬 Example of NOT in 1NF:

- → After 1NF, each treatment becomes a separate row.

Step 2: Second Normal Form (2NF)

Rule:

- Must be in 1NF, and
- No partial dependency — every non-key attribute depends on the entire primary key.

Let's assume primary key = (Patient_ID, Doctor_Name) (composite).

- But Doctor's phone depends only on Doctor_Name — not Patient_ID.

👉 Partial dependency detected!

- So, we split the table.
- Now, no partial dependencies — each non-key attribute depends fully on the primary key.

Patient_ID	Patient_Name	Treatment	Room_No	Doctor_Name
P101	Arun	Fever	R101	Dr. Kiran
P102	Meena	Cold	R102	Dr. Kiran
P103	Kavi	Fracture	R103	Dr. Ravi

Doctor_Name	Doctor_Phone
Dr. Kiran	9876543210
Dr. Ravi	9786542109

Step 3: Third Normal Form (3NF)

Rule:

- Must be in 2NF, and
- No transitive dependency (non-key attribute depending on another non-key).
 - In Table Patient, Room_No might depend on Treatment (say all “Fever” patients go to R101). That’s transitive dependency.

Step 3: Third Normal Form (3NF)

Patient_ID	Patient_Name	Doctor_Name	Treatment_ID
P101	Arun	Dr. Kiran	T01
P102	Meena	Dr. Kiran	T02
P103	Kavi	Dr. Ravi	T03

Doctor_Name	Doctor_Phone
Dr. Kiran	9876543210
Dr. Ravi	9786542109

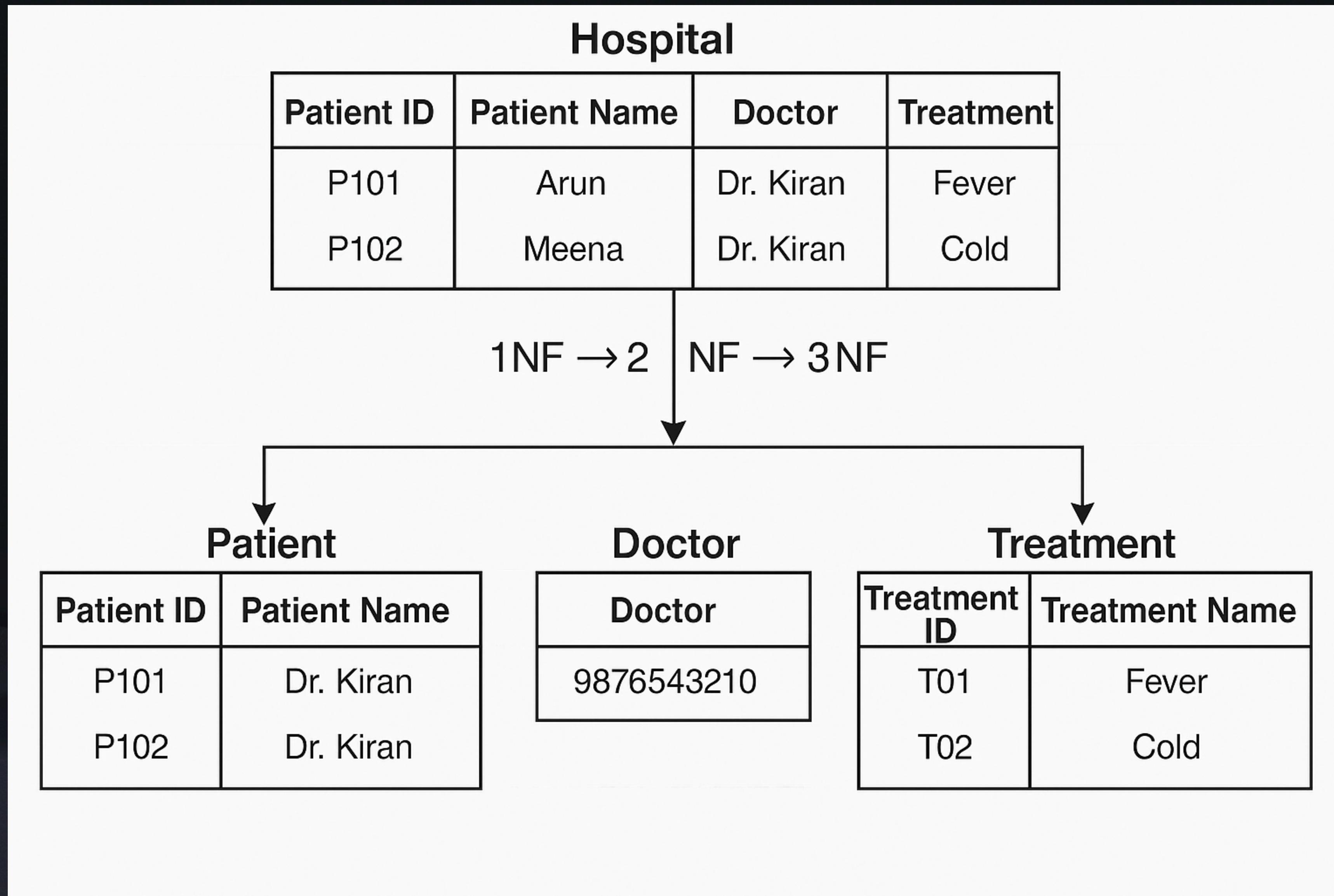
Treatment_ID	Treatment_Name	Room_No
T01	Fever	R101
T02	Cold	R102
T03	Fracture	R103

Final 3NF Schema (Hospital DB)

- Data redundancy eliminated
- Easy to update Doctor/Room separately
- Perfectly relational

Table	Primary Key	Purpose
Patient	Patient_ID	Stores patient personal info
Doctor	Doctor_Name	Stores doctor contact info
Treatment	Treatment_ID	Stores treatment details

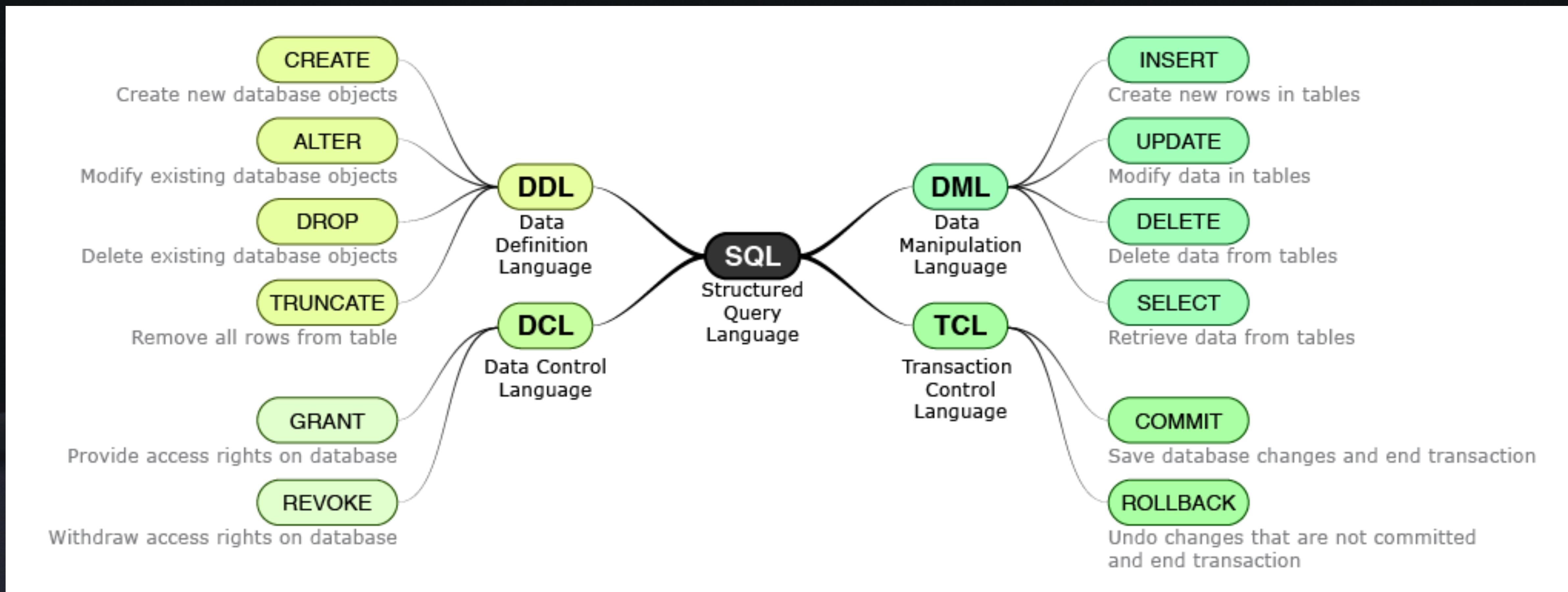
Hospital Database Normalization ER Diagram



Mini Student Task

Emp_ID	Emp_Name	Dept_Name	Dept_Loc	Manager_Name

Introduction to SQL



Types of SQL Operators

Category	Description
Arithmetic Operators	Perform mathematical operations
Comparison Operators	Compare one value with another
Logical Operators	Combine multiple conditions
Range / Matching Operators	Check for specific ranges or patterns
Set Operators	Combine results from multiple queries
Special Operators	Miscellaneous (IS NULL, EXISTS, ANY, ALL, etc.)

Arithmetic Operators

```
SELECT Product_Name, Price, Stock, Price * Stock AS Total_Value
```

```
FROM Product;
```

Operator	Description	Example
+	Addition	Price + 500
-	Subtraction	Stock - 2
*	Multiplication	Price * Quantity
/	Division	Price / 2
%	Modulus	Stock % 2

Comparison Operators

```
SELECT Product_Name, Price  
FROM Product  
WHERE Price > 10000;
```

Operator	Description	Example
=	Equal to	City = 'Chennai'
<> or !=	Not equal to	Category_ID <> 2
>	Greater than	Price > 10000
<	Less than	Stock < 10
>=	Greater than or equal to	Rating >= 4
<=	Less than or equal to	Stock <= 5

Logical Operators

SELECT Name, City

FROM Customer

WHERE City = 'Chennai' OR City = 'Madurai';

Operator	Description	Example
AND	Returns TRUE if <i>all</i> conditions are true	Price > 5000 AND Stock > 10
OR	Returns TRUE if <i>any</i> condition is true	City = 'Chennai' OR City = 'Madurai'
NOT	Reverses the result	NOT (City = 'Chennai')

Range Operators

a) BETWEEN

Used to filter within a range (inclusive).

```
SELECT Product_Name, Price
```

```
FROM Product
```

```
WHERE Price BETWEEN 1000 AND  
5000;
```

 Shows all products priced between ₹1,000 and ₹5,000.

b) IN

Used to test whether a value matches any value in a list.

```
SELECT Name, City
```

```
FROM Customer
```

```
WHERE City IN ('Chennai', 'Madurai',  
'Coimbatore');
```

 Lists customers from the given cities.

c) NOT IN

Inverse of IN – excludes matches.

```
SELECT Name, City
```

```
FROM Customer
```

```
WHERE City NOT IN ('Chennai',  
'Madurai');
```

 Lists customers who are not from Chennai or Madurai.

Pattern Matching Operator — LIKE

Used for partial string matches.

💡 Examples:

SELECT Name FROM Customer WHERE Name LIKE 'A%';

Names starting with "A".

SELECT Email FROM Customer WHERE Email LIKE '%gmail.com';

All customers using Gmail.

SELECT Product_Name FROM Product WHERE Product_Name LIKE '_a%';

Product names where 2nd letter is 'a'.

Wildcard	Description
%	Matches zero or more characters
_	Matches exactly one character

NULL Operators

a) IS NULL

Finds rows where a value is NULL.

```
SELECT Name, Pincode
```

```
FROM Customer
```

```
WHERE Pincode IS NULL;
```

 Lists customers without a Pincode.

b) IS NOT NULL

Finds rows where a value exists.

```
SELECT Name
```

```
FROM Customer
```

```
WHERE Phone IS NOT NULL;
```

 Lists customers who have provided phone numbers.

Special Operators

a) EXISTS

Checks if a subquery returns any rows.

```
SELECT Name
```

```
FROM Customer c
```

```
WHERE EXISTS (
```

```
    SELECT 1 FROM Orders o WHERE  
    o.Customer_ID = c.Customer_ID
```

```
);
```

✓ Returns only customers who have placed at least one order.

b) ANY

Compares a value to any value returned by a subquery.

```
SELECT Product_Name, Price
```

```
FROM Product
```

```
WHERE Price > ANY (SELECT Price  
FROM Product WHERE Category_ID =  
2);
```

✓ Shows products costlier than at least one item in Category 2.

c) ALL

Compares a value to all values returned by a subquery.

```
SELECT Product_Name, Price
```

```
FROM Product
```

```
WHERE Price > ALL (SELECT Price  
FROM Product WHERE Category_ID =  
2);
```

✓ Shows products costlier than every item in Category 2.

Set Operators

SELECT City FROM Customer

UNION

SELECT City FROM Supplier;

Operator	Description	Removes Duplicates?
UNION	Combines results	<input checked="" type="checkbox"/> Removes duplicates
UNION ALL	Combines results	<input type="checkbox"/> Keeps duplicates
INTERSECT	Returns common results	<input checked="" type="checkbox"/>
EXCEPT / MINUS	Returns rows in first query but not in second	<input checked="" type="checkbox"/>

Assignment Operator

UPDATE Product

SET Stock = Stock + 5

WHERE Product_Name = 'Laptop';

Operator	Description	Example
=	Assigns a value to a column	SET Stock = Stock + 10

Concatenation Operator

```
SELECT CONCAT(Name, ' - ', City) AS CustomerInfo  
FROM Customer;
```

SQL Joins - Inner Join

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

```
SELECT c.Customer_ID, c.Customer_Name, c.City, o.Order_ID, o.Product,  
o.Amount  
FROM Customer c  
INNER JOIN Orders o  
ON c.Customer_ID = o.Customer_ID;
```

Customer_ID	Customer_Name	City	Order_ID	Product	Amount
C1	Arjun	Chennai	O1	Laptop	55000
C1	Arjun	Chennai	O2	Mouse	800
C2	Meena	Madurai	O3	Chair	2500

Order_ID	Customer_ID	Product	Amount
O1	C1	Laptop	55000
O2	C1	Mouse	800
O3	C2	Chair	2500
O4	C5	Headphones	1200

Customer_ID	Customer_Name	City
C1	Arjun	Chennai
C2	Meena	Madurai
C3	Ravi	Coimbatore
C4	Priya	Salem

SQL Joins - Left Join

- Returns all records from the left table (Customer),
- and the matching rows from the right table (Orders).
- If no match → NULL on right side.

```

SELECT c.Customer_ID, c.Customer_Name, c.City, o.Order_ID, o.Product,
o.Amount
FROM Customer c
LEFT JOIN Orders o
ON c.Customer_ID = o.Customer_ID;
    
```

Customer_ID	Customer_Name	City	Order_ID	Product	Amount
C1	Arjun	Chennai	O1	Laptop	55000
C1	Arjun	Chennai	O2	Mouse	800
C2	Meena	Madurai	O3	Chair	2500
C3	Ravi	Coimbatore	NULL	NULL	NULL
C4	Priya	Salem	NULL	NULL	NULL

Order_ID	Customer_ID	Product	Amount
O1	C1	Laptop	55000
O2	C1	Mouse	800
O3	C2	Chair	2500
O4	C5	Headphones	1200

Customer_ID	Customer_Name	City
C1	Arjun	Chennai
C2	Meena	Madurai
C3	Ravi	Coimbatore
C4	Priya	Salem

SQL Joins - Full Outer Join

- Returns all records when there's a match in either left or right table.
- Non-matching rows show NULLs on respective sides.

```

SELECT c.Customer_ID, c.Customer_Name, c.City, o.Order_ID, o.Product, o.Amount
FROM Customer c
LEFT JOIN Orders o ON c.Customer_ID = o.Customer_ID
UNION
SELECT c.Customer_ID, c.Customer_Name, c.City, o.Order_ID, o.Product, o.Amount
FROM Customer c
RIGHT JOIN Orders o ON c.Customer_ID = o.Customer_ID;
    
```

Customer_ID	Customer_Name	City	Order_ID	Product	Amount
C1	Arjun	Chennai	O1	Laptop	55000
C1	Arjun	Chennai	O2	Mouse	800
C2	Meena	Madurai	O3	Chair	2500
C3	Ravi	Coimbatore	NULL	NULL	NULL
C4	Priya	Salem	NULL	NULL	NULL
NULL	NULL	NULL	O4	Headphones	1200

Order_ID	Customer_ID	Product	Amount
O1	C1	Laptop	55000
O2	C1	Mouse	800
O3	C2	Chair	2500
O4	C5	Headphones	1200

Customer_ID	Customer_Name	City
C1	Arjun	Chennai
C2	Meena	Madurai
C3	Ravi	Coimbatore
C4	Priya	Salem

SQL Joins - Cross Join

- Returns the Cartesian product of both tables –
- each row from left table combines with all rows from right table.

```
SELECT c.Customer_Name, o.Product  
FROM Customer c  
CROSS JOIN Orders o;
```

Customer_Name	Product
Arjun	Laptop
Arjun	Mouse
Arjun	Chair
Arjun	Headphones
.....

Order_ID	Customer_ID	Product	Amount
O1	C1	Laptop	55000
O2	C1	Mouse	800
O3	C2	Chair	2500
O4	C5	Headphones	1200

Customer_ID	Customer_Name	City
C1	Arjun	Chennai
C2	Meena	Madurai
C3	Ravi	Coimbatore
C4	Priya	Salem

Thank You