Database Management Systems Lab Manual

CPE 353L

Fall 2026



University of Management & Technology

School *of* Engineering

Department of Electrical Engineering

Name : _____

Student ID : _____

Section : _____

University of Management and Technology

School of Engineering
Department of Electrical Engineering

Course Outline (Fall 2026)

Course Code: **CPE-353L** Course Title: **Database Management Systems Lab**

Program	BS-Computer Engineering
Credit Hours	1
Contact Hours	3
Knowledge Domain	Engineering Foundation
Duration	One semester
Prerequisites	Nil
Resource Person	M.Annas Marwat
Counseling Timing	See Office Window (SEN-310)
Contact	muhammad.annas@umt.edu.pk

Chairman/Director signature…….

Dean's signature…… **Date……**

Learning Objective

This lab offers a practical foundation in database systems guiding students from basic concepts of data and DBMS to the design and implementation of real-world database applications. Students will create ER diagrams, convert them into relational models and apply SQL commands for defining, manipulating and querying data. Through hands-on exercises, they will work with DDL, DML, functions, operators, constraints, relationships and joins. Advanced topics such as grouping, sorting, indexing, subqueries and views will also be practiced to improve performance and efficiency. By the end students will be able to design, implement and optimize relational databases using MySQL.

Learning Methodology

Practical, interactive, participative

Grade Evaluation Criteria

Following are the criteria for the distribution of marks to evaluate final grade in the semester.

Marks Evaluation	**Marks**
Lab Manuals & Performance:	50
Open-Ended Lab	10
Final Viva + Final Performance	20 +20
Total:	**100**

Recommended Textbooks

- "Database System Concepts" by Silberschatz, Korth, and Sudarshan
- "Fundamentals of Database Systems" by Elmasri and Navathe

Reference Books

1) "SQL: The Complete Reference" by Groff and Weinberg

2) "NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence" by Sadalage and Fowler

Course Learning Outcomes (CLOs)

CLO 1 : Follow standard procedures to design, implement and optimize relational databases with ER diagrams and SQL, delivering concrete database results from practical work.

CLO 2 : Explain the relationship between database design decisions and system performance through experimental investigation and analysis.

CLO 3: Complete database design, implementation and optimization tasks effectively using appropriate tools.

CLO 4: Demonstrate the importance of professional ethics and responsibilities in database design.

Relation to EE Program Learning Outcomes (PLOs)

CLOs	Related PLOs	Levels of Learning	Teaching Methods	CLO Attainment checked in
CLO 1	PLO 5	Psy-3	Instructions, Lab Work	Lab Sessions, Exam
CLO 2	PLO 4	Cog-3	Instructions, Lab Work	Lab Sessions (Viva), Exam
CLO 3	PLO 10	Aff-3	Instructions, Lab Work	Lab Sessions, Manuals, Exam
CLO 4	PLO 8	Aff-3	Instructions, Lab Work	Lab Sessions, Manuals

Calendar of Course contents to be covered during semester

Course Code: **IET-217L** Course title: **Database Systems Lab**

Week	Course Contents / Lab Experiments	CLOs
1	Implement Database Systems and DBMS Architecture	1,3
2	Develop Data Model using Entity Relationship (ER) Diagram	1,2,3,4
3	Construct Database Schema based on SQL Modeling	1,2,3,4

4	Implementation of DDL commands of SQL ● Create table ● Alter table ● Drop Table	1,2,3,4
5	Implementation of DML & DQL commands of SQL with suitable examples ● Insert ● Update ● Delete ● Select, From & Where	1,2,3,4
6	Implementation of different types of function	1,2,3,4
7	Implementation of different types of operators in SQL	1,2,3,4
	Mid Term Week	
8	Implementation of MySQL Constraints and Relationships	1,2,3,4
9	Implementation of different types of Joins	1,2,3,4
10	Implementation of Group By having clause & Order by clause	1,2,3,4
11	**Open-Ended Lab**	1,2,3,4
12	Implementation of CASE Statements in SQL for Conditional Logic	1,2,3,4
13	Implementation of Indexing	1,2,3,4

14	Implementation of Sub queries & Views	1,2,3,4

Complex Engineering Problem/Activity

Complex Engineering Problem Details	Included: No
Open Ended Lab	Included: Yes

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric												
Section : I												
Areas Assessed		Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
		(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)		Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)		Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)		Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr #	Maximum Marks ►		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Introduction to Databases & DBMS

OBJECTIVES

1. Introduction to Database.
2. Understanding the Evolution History and Different Types of Database.
3. Introduction to DBMS.
4. Introduction to Meta Data.

OUTCOMES

1. To understand the concepts of Data, Database, and DBMS.
2. To learn what a Schema and ER Diagram are.
3. To manually design a simple database schema and ER diagram

THEORETICAL BACKGROUND

Database

A database is an organized collection of structured information or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to the just database.

Evolution of the Database

Databases have evolved dramatically since their inception in the early 1960s. Navigational databases such as the hierarchical database (which relied on a tree-like model and allowed only a one-to-many relationship), and the network database (a more flexible model that allowed multiple relationships), were the original systems used to store and manipulate data. Although simple, these early systems were inflexible. In the 1980s, relational databases became popular, followed by object-oriented databases in the 1990s. More recently, NoSQL databases came about as a response to the growth of the internet and the need for faster speed and processing of unstructured data. Today, cloud databases and self-driving databases are breaking new ground when it comes to how data is collected, stored, managed, and utilized.

Types of Databases

There are many different types of databases. The best database for a specific organization depends on how the organization intends to use the data.

1. **Relational Databases**

   Relational databases became dominant in the 1980s. Items in a relational database are organized as a set of tables with columns and rows. Relational database technology provides the most efficient and flexible way to access structured information.

2. **Object-Oriented Databases**

   Information in an object-oriented database is represented in the form of objects, as in object- oriented programming.

3. **Distributed Databases**

   A distributed database consists of two or more files located in different sites. The database may be stored on multiple computers, located in the same physical location, or scattered over different networks.

4. **Data Warehouses**

   A central repository for data, a data warehouse is a type of database specifically designed for fast query and analysis.

5. **NoSQL Databases**

   A NoSQL, or nonrelational database, allows unstructured and semi structured data to be stored and manipulated (in contrast to a relational database, which defines how all data inserted into the database must be composed). NoSQL databases grew popular as web applications became more common and more complex.

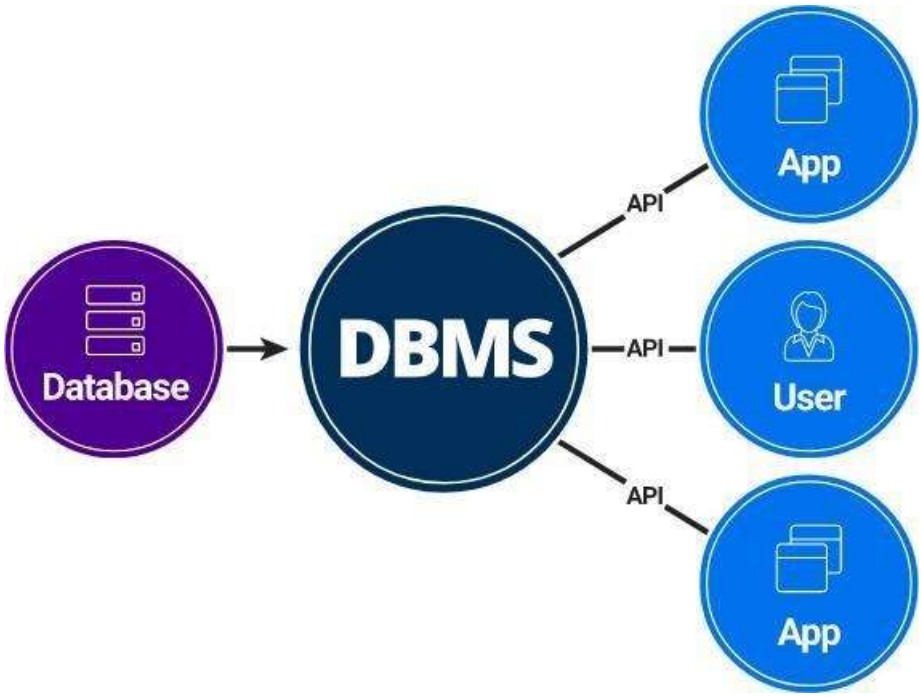
6. **Graph Databases**

   A graph database stores data in terms of entities and the relationships between entities.

Database Management System (DBMS)

A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database. A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure.

STRUCTURE OF DATABASE & DBMS



MAIN FUNCTION OF DBMS

- Data storage and retrieval
- Controlled access to data
- Data integrity and consistency
- Reduction in redundancy

META DATA

Metadata means ***data about data*.** It describes the structure, meaning and format of the actual data stored in a database. Metadata tells us **what kind of data** is being stored, **how** it is stored and **what it represents**.

Explanation:

When we create a database, we store many kinds of data names, IDs, numbers, etc. But to understand that data properly, we also need ***information about the data itself,*** for example its **type**, **length**, **format**, or **meaning**. That descriptive information is called **metadata**.

Think of metadata as a "label" or "description" that helps you understand your data.

Real Life Scenario:

The metadata will include:

Attribute	Data Type	Size	Description
ID	Integer	4	Unique ID of student
Name	Varchar	50	Student's full name
Dept	Varchar	10	Department code
GPA	Float	3	Student's GPA

All this information **about columns and their structure** is metadata.

MetaData In terms of DBMS:

In a DBMS, metadata is stored in **system tables**.

- Table names
- Column names
- Data types
- Relationships between tables
- Constraints (like Primary Key, Foreign Key)

Why Metadata is Important:

1. It describes the structure of the database.
2. It helps DBMS to interpret and process the data correctly.

3. It improves data organization and reduces confusion.
4. It supports **data independence** — users can query data without knowing how it is physically stored.

SCHEMA

A **Schema** defines the **structure** of a database; it shows **how data is organized**, how **tables are connected**, and what **relationships** exist between them. It acts as a **blueprint** or **layout** of the entire database.

Explanation:

Just like an architect designs a building's plan before construction, a **schema** is the *plan of the database* before actual data is inserted. It tells:

- How many **tables** will be created
- What **attributes (columns)** each table will have
- What **keys** (Primary, Foreign) will link them together

In DBMS, the schema is defined using **DDL commands** (like CREATE TABLE). It helps maintain **data consistency** and avoids confusion when multiple people work on the same database.

Explanation of Relationships:

- Each **Student** belongs to one **Department** → *(Dept_ID is a foreign key in Student table)*
- Each **Course** also belongs to one **Department** → *(Dept_ID is a foreign key in Course table)*
- Therefore, **the Department** is the *main table* that connects both Student and Course tables.

Example Data:

Student Table

Student_ID	Name	Dept_ID
1	Ali	D01
2	Sara	D02

Department Table

Dept_ID	Dept_Name
D01	Computer Engineering
D02	Electrical Engineering

Course Table

Course_ID	Course_Name
C01	Database Systems
C02	Circuit Analysis

ER DIAGRAM (ENTITY RELATION DIAGRAM)

An **ER Diagram** is a visual representation of entities and their relationships in a database. It helps in designing the database before creating it.

Basic Components of ER Diagram:

- **Entity:** A real-world object (e.g., Student, Department, Course) — drawn as a rectangle.
- **Attribute:** Properties of an entity (e.g., Name, RollNo) — drawn as ovals.
- **Relationship:** Connection between entities (e.g., Student *enrolled in* Course) — drawn as a diamond.

Scenario:

A university wants to maintain student records. Each student belongs to one department and each department offers many courses. From this, we can define:

Entities:

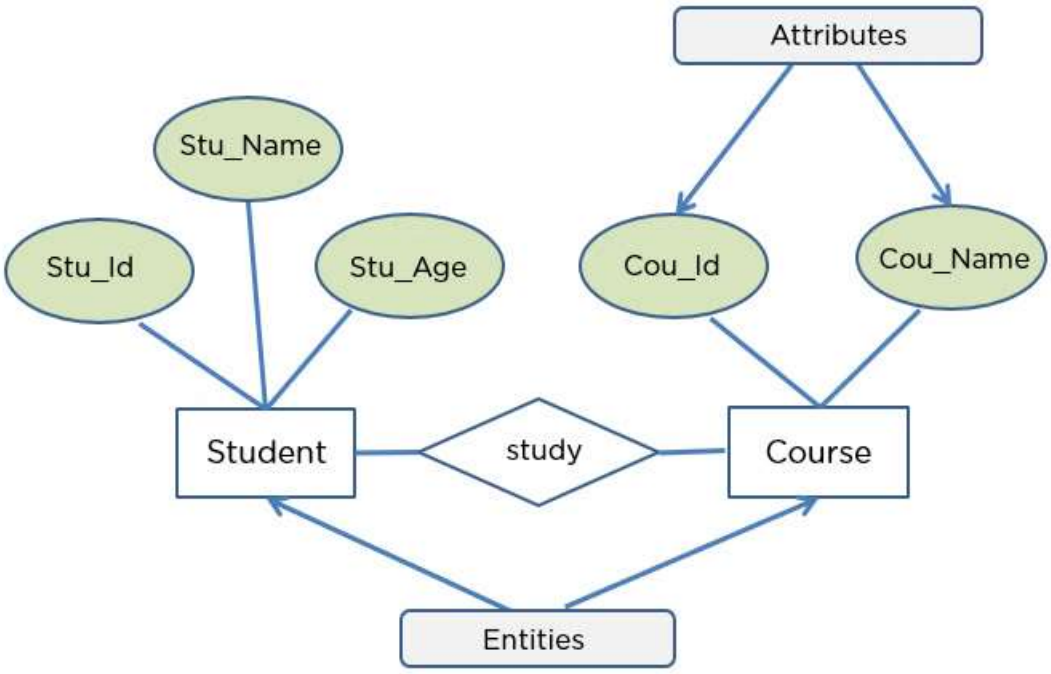
- Student
- Course

Relationships:

- Student study this Course

Attributes:

- Student: Student_ID, Name, Age
- Course: Course_ID, Course_Name



LAB TASKS

Task 1

The university maintains a list of students. For every student the system stores (Name, Department , Registration No., Enrolled Year , Contact No)

Task Perform:

1. Identify Data, Information, and Metadata from this scenario.
2. Prepare a Metadata Table describing the attributes of the ***Student*** **table**.

Expected Outcome:

Data: Ali, CE, 2023-CE-07, 03001234567 raw stored facts

Information: "Name (Reg No.) is a dept of student admitted in 2023. Contact no is 0300……. ."

Metadata: Describes what each field means, its data type and format.

Task 2

To identify entities, attributes, and relationships from a real-world business scenario and represent them in the form of an Entity-Relationship (ER) model.

Description:

A Retail Management System is being designed for a chain of electronic stores. The company wants to maintain records of its Stores, Employees, Customers, Products, Suppliers, and Sales Orders in a database.

Requirement:

- Each Store has a unique Store ID, Name, Location, and Manager.
- Each Employee works in one store, has an Employee ID, Name, Position, Salary and Contact Number.
- Each Customer has a Customer ID, Name, Contact Number and City.
- Each Product has a Product ID, Name, Category, Unit Price and is supplied by one Supplier.

- Each Supplier has a Supplier ID, Company Name, Contact Person and Phone.
- Each Sales Order has an Order ID, Date, and Total Amount. Every order is placed by one Customer, handled by one Employee and can contain multiple Products.

Task Performed:

Students must carefully read the scenario and:

1. Identify Entities and their Attributes.
2. Mention the Primary Key (PK) for each entity.
3. Determine the Relationships between entities **[Hint : Relationship (Employee → Store), Type (Works In), Description (Each employee works in one store)]**

Task 3

A company manages its projects using a database. The database keeps information about Employees, Departments, Projects and Clients.

Description:

- Each Employee has an ID, Name, Email, Contact Number and works in one Department.
- Each Department has a unique ID, Name and Location.
- Each Project has a Project ID, Title, Budget and is handled by one Department.
- Each Client has a Client ID, Name, Contact Person and the Project they are associated with.

Activity:

1. Identify Entities and their Attributes.
2. Mention the Primary Key (PK) for each entity.
3. Find relationships between entities (like Works In, Handles, Assigned To).

Hint Relationships:

- Employee → Department → (Works In)
- Project → Department → (Handled By)
- Client → Project → (Assigned To)

Expected Outcome:

Students will identify multiple entities, their relationships and understand how to design an ER model for a real-world organization.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric													
Lab # 1		Lab Title : Implement Database Systems and DBMS Architecture					Section : I						
Areas Assessed		Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total					
		(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)						
Poor (0 to 30%)		Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork						Total
Satisfactory (31 to 70%)		Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork	CLO 1	CLO 2	CLO 3	CLO 4		
Very Good (71 to 100%)		Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work						
Sr#	Maximum Marks ▶		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Introduction to ER Diagram

OBJECTIVES

1. To understand the basic concepts of Entity Relationship Diagrams (ERDs).
2. To identify entities, attributes and relationships among data objects.
3. To learn how to represent data relationships using ERD symbols.
4. To design a simple ER diagram for a given real-world scenario.
5. To prepare students for logical database design through ER modeling.

OUTCOMES

1. Explain the basic concepts and components of an ER diagram.
2. Identify entities, attributes, and relationships in a given problem statement.
3. Use appropriate symbols to represent ERD components.
4. Construct a complete ER diagram for a real-world scenario.
5. Relate ERD concepts to the logical structure of a database.

THEORETICAL BACKGROUND

ER Diagram

An **Entity Relationship (ER) Diagram** is a simple diagram that shows how different things (called *entities*) are related to each other in a system. for example, *students, teachers* or *courses.* ER diagrams are mostly used to **design or understand databases** in the field of Software Engineering.

They use some common symbols:

- **Rectangles** for entities,
- **Diamonds** for relationships,
- **Ovals** for attributes,
- **lines** to connect them.

In short, entities act like **nouns** (Student, Teacher) and relationships act like **verbs** (Enrolls, Teaches).

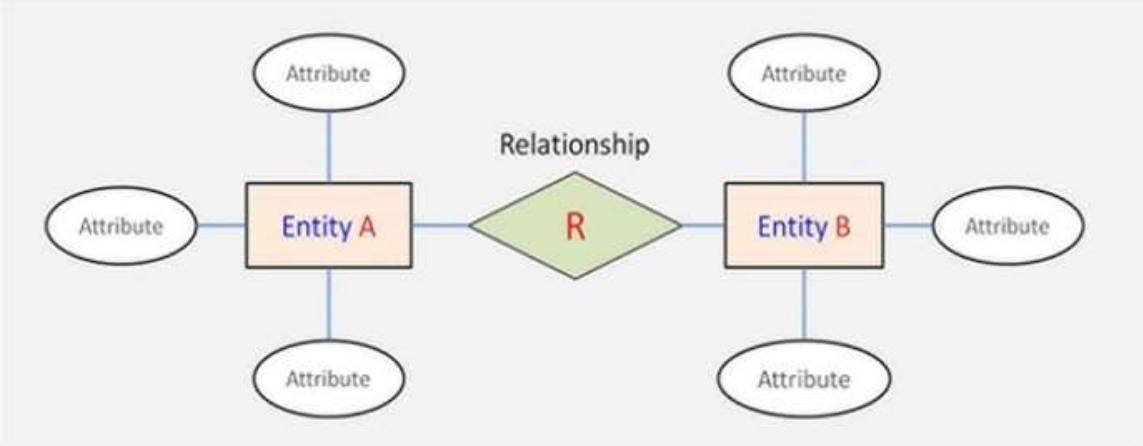


Fig 1.1 ER Diagram

IMPORTANCE OF ER DIAGRAM

1. ER Diagrams help to **visually understand** how data is connected in a system.
2. They make it **easier to plan and design databases** before actual implementation.
3. They help **identify entities, attributes, and relationships** clearly.
4. ERDs help **avoid design mistakes** by showing the complete data structure early.
5. They act as a **blueprint for creating database tables** and setting relationships.
6. ER Diagrams improve **communication between developers, analysts, and clients** by providing a clear picture of the system.

Uses Of Entity Relation Diagram

1. **Database Design:**
   ER diagrams are used to plan and design databases. They help in understanding what data is needed, how it is related, and how it should be stored. In software engineering, an ER diagram is the first step before creating tables in a database.
2. **Database Troubleshooting:**
   ER diagrams help to check existing databases and find mistakes in design or relationships. By looking at the diagram, we can easily see where the problem is.

3. **Business Information Systems:**
   ER diagrams are used in businesses to design databases for managing daily operations such as employees, customers, and products. This helps to make processes faster and improves data accuracy.

4. **Business Process Re-engineering (BPR):**
   ER diagrams are used when a company wants to improve or redesign its business processes. They help in planning new database systems that match updated workflows.

5. **Education:**
   ER diagrams are useful for schools and universities to plan data storage — such as student records, courses, and attendance systems. They make it easier to organize and retrieve educational information.

6. **Research:**
   In research projects, ER diagrams help to design structured databases for storing and analyzing collected data in an organized way.

BASIC COMPONENT OF ER DIAGRAM

1. Entity
2. Attribute
3. Relationship
4. Keys (Primary, Foreign)
5. Cardinality

Entity

An **Entity** is anything that has data stored about it. It can be a **person, object, place, event or concept** that we want to keep information about.

Example:

- Student
- Teacher
- Course
- Car
- Product

In an ER diagram, an **Entity** is shown by a **rectangle** and its name is written inside it. Think of an entity as



a **noun, something** that exists and we can describe.

Entity Type

An Entity Type is a group or category of similar entities.
For example: Student is an entity type, and Ali, Sara, and Usman are specific students (instances of that type).

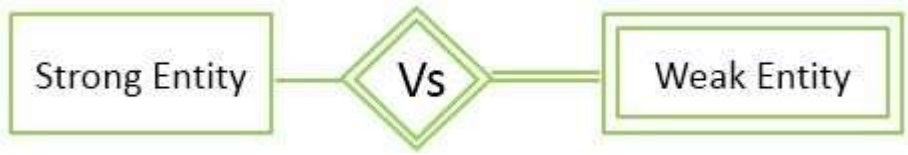
Entity Set

An **Entity Set** is a collection of entities of the same type at a given time.
For example: All students enrolled this semester make up a **Student Entity Set**.

Types of Entities

1. **Strong Entity:**
    - Exists on its own (independent).
    - Example: *Student, Teacher, Department*
    - Represented by a **single rectangle**.
2. **Weak Entity:**
    - Depends on another entity for its existence.
    - Example: *Exam Result* depends on *Student or Course* .
    - Represented by a **double rectangle** and connected with a **double diamond** relationship.

A **Weak Entity** cannot exist **without a strong entity**.



Attribute

An **Attribute** is a **property or characteristic** of an entity. It gives **more details** about the entity.

Example:
If the entity is **Student**, its attributes can be:

- Student_ID
- Name
- Email
- Age

In an ER diagram attributes are shown by **ovals** connected to their entity (rectangle).

Simple way to remember:
Attributes are the **details** that describe an entity.

Types of Attributes

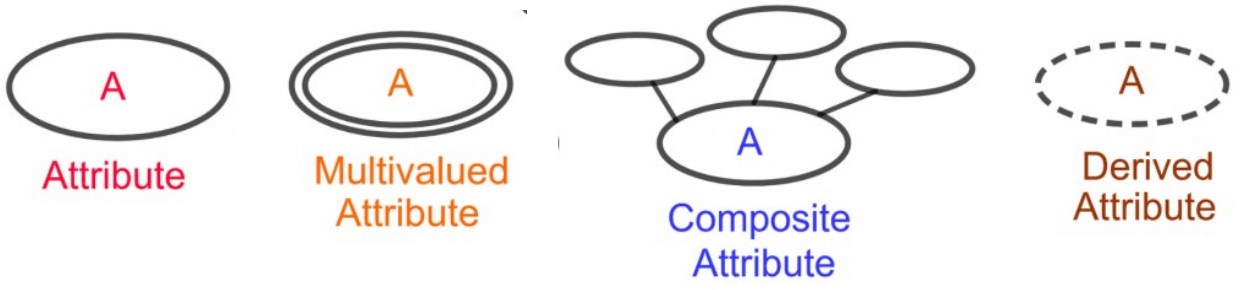
1. **Simple Attribute:**
    - A single value that cannot be divided further.
    - **Example:** Name, Age, Roll_No
    - **Representation:** Shown by a **single oval**
2. **Composite Attribute:**
    - An attribute that can be **divided into smaller parts.**
    - **Example:** Full_Name → (First_Name, Last_Name)
      Address → (City, Street, Zip_Code)
    - **Representation:** Shown by a **main oval** connected to **sub-ovals**.
3. **Derived Attribute:**
    - An attribute whose value is **calculated from another attribute.**
    - **Example:** Age (derived from Date_of_Birth). Because age varies but date of birth remains same
    - **Representation:** Shown by a **dotted oval**

4. Single-Valued Attribute:

- Has **only one value** for each entity.
- **Example:** CNIC, Roll_No
- **Representation:** Shown by a **single oval**

5. Multi-Valued Attribute:

- Can have **more than one value** for an entity.
- **Example:** Phone_Numbers, Email_Addresses.Because may be a person have 2 or more Phone No. or Email Address
- **Representation:** Shown by a **double oval** (◎)



Relationship

A **relationship** shows **how two or more entities are connected** or **interact** with each other. You can think of it as a **verb (action)** between entities.

Example:

- A Student ***enrolls*** *in* a Course
- A Teacher ***teaches*** a Course
- A Department ***offers*** a Program

In an ER diagram, a relationship is shown by a **diamond** shape. The name of the relationship (like *enrolls*, *teaches*, *offers*) is written inside the diamond.

Types Of Relationship

1. **Weak Relationship**
   A Weak Relationship connects a **weak entity** to its owning strong entity. It shows that the weak entity depends on the strong entity for its existence.
   Example:

   Student (Strong Entity) & Fee_Record (Weak Entity)
   A *Fee_Record* cannot exist without a *Student*. If the student is deleted, the fee record has no meaning. So, the relationship between them *"Has"* is a **Weak Relationship**.

2. **Recursive Relationship**
   When the same entity participates more than once in a relationship.
   Example:
   An Employee *supervises* another Employee.
   So, both sides are the same entity (*Employee supervises Employee*).
   Representation: (Employee) —◇— (Employee)





Keys

Keys are special attributes (or a set of attributes) that help **uniquely identify** records in a table and **create relationships** between different tables in a database. Keys ensure that **data is unique, accurate and linked properly** within the database.

Types of Keys:

1. **Primary Key**

   The **main key** is chosen from candidate keys to uniquely identify each record in a table.
   Example: Student ID in Student table.

2. **Foreign Key**

   A key that **connects one table to another**. It is a **primary key** in another table.
   Example: Dept_ID in Student table refers to the Dept_ID in Department table. Dept_ID in Department Table is Primary Key

Cardinality

Cardinality shows **how many instances of one entity are related** to instances of another entity. It tells us the **number of possible relationships** between entities.

For Example: *one student can enroll in many courses.*

Types of Cardinality

1. **One-to-One (1:1)**
   Each record in one entity is connected to only one record in another entity.
   Example: One Student has one Student ID.



2. **One-to-Many (1:N)**

   One record in the first entity is connected to **many** records in the second entity.
   Example: One **Teacher** teaches **many Students**.



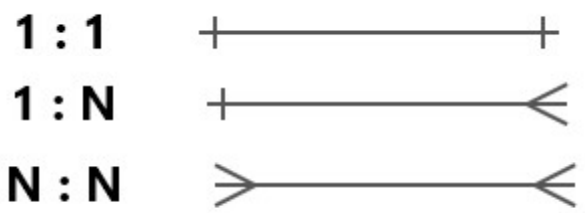
3. **Many-to-Many (M:N)**

   Many records in one entity can be related to **many** records in another entity.
   Example: Many **Students** can enroll in **many Courses**.



4. **Minimum–Maximum Cardinality**
   Sometimes, we also show whether a relationship is **0** to **1 and 0** to **Many.**
   - **Zero-to-One:**
     An entity may or may not be related to another.
     Example: A **Student** *may have* **0 or 1 Scholarship**.

- **Zero-to-Many:**
  An entity may have **no related records** or **many related records**.
  Example: A **Teacher** *may teach no class* or *many classes*.



How To Draw a Basic ER DIAGRAM

Following Are basic method to draw ER Diagram

- Purpose and Scope
- Entity
- Attributes
- Relations
- Cardinality

LAB TASKS

Task 1

School Management System

Question:
A school maintains records of its **Students, Teachers, and Classes.**

- Each student has a **Student_ID, Name, Roll_No, Age and Contact_No.**
- Each teacher has a **Teacher_ID, Name, Subject and Qualification.**
- Each class has a **Class_ID, Class_Name, Section and Room_No.**

Instructions:

1. Identify all **Entities** and their **Attributes (Mention which attribute used).**
2. Mention the **Primary Key (PK)** for each entity.
3. Identify possible **Relationships** among these entities.
4. **Draw an ER Diagram** for the given scenario.
5. Clearly define **Cardinalities (1–1, 1–M, M–M)** for each relationship.

Task 2

Course Enrollment System

Question:
A university keeps records of **Students** and **Courses** through an **Enrollment** system.

- Each student can register for one or more courses.
- Each student has **Student_ID, Name, Email and Contact_No.**
- Each course has **Course_ID, Title and Credit_Hours.**
- Each enrollment record has **Enrollment_No, Date and Grade.**

Instructions:

1. Identify all **Entities, Attributes and Keys**.
2. Point out if any **Weak Entity** exists and explain why.
3. Establish relationships such as *Enrolls In* or *Registers For*.
4. **Draw a neat ER Diagram** showing entities, attributes and relationships.
5. Indicate **Cardinalities** clearly on the diagram.

Task 3

Hospital Management System

Question:

A hospital keeps information about its **Patients, Doctors, Appointments and Departments.**

- Each patient has **Patient_ID, Name, Age, Address and Phone.**
- Each doctor has **Doctor_ID, Name, Specialization and Phone.**
- Each appointment includes **App_ID, Date, Time and Remarks.**
- Each department has **Dept_ID, Dept_Name and Location.**

Instructions:

1. Identify all **Entities**, **Attributes** and their **Keys**.
2. Specify which entities are **Strong** or **Weak.**

3. Identify meaningful **Relationships** (e.g., “Patient Books Appointment”, “Doctor Belongs To Department”).
4. **Draw the ER Diagram** with relationships, attributes and entities properly connected.
5. Label all **Cardinalities** accurately.

Task 4

University Management System

Question:

A university maintains detailed information about its **Students, Courses, Instructors, Departments, Hostels and Fees.**

- Each student has **Student_ID, Name, Gender, Date_of_Birth, Age (derived), Address and Contact_No.**
- Each course has **Course_ID, Title, Credit_Hours and Duration.**
- Each instructor has **Instructor_ID, Name, Qualification and Contact_No.**
- Each department has **Dept_ID, Dept_Name and Location.**
- Each hostel has **Hostel_ID, Hostel_Name and Room_No.**
- Each fee record has **Fee_ID, Amount, Due_Date and Status.**

Instructions:

1. Identify all **Entities**, **Attributes** and **Derived Attributes.**
2. Mention **Primary Keys**, **Foreign Keys** and any **Weak Entities**.
3. Find all possible **Relationships** among entities.
4. **Draw a complete ER Diagram** including entities, attributes, keys and relationships.
5. Define **Cardinalities** and ensure logical consistency throughout the model.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric

Lab # 2 **Lab Title :** Develop Data Model using Entity Relationship (ER) Diagram **Section :** I

Areas Assessed		Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
		(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)		Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)		Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)		Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
S#	Maximum Marks ►	2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼ Student Name ▼											
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

Construct Database Schema based on SQL Modeling

OBJECTIVES

1. To introduce students to the fundamental concepts of SQL (Structured Query Language).
2. To develop an understanding of how SQL is used to create and manage databases.
3. To familiarize students with SQL commands for data definition, manipulation, and querying.
4. To enable students to interact with databases effectively using SQL statements.

OUTCOMES

1. Understand the concept and purpose of SQL.
2. Learn different types of SQL commands such as DDL, DML, DQL, DCL, and TCL.
3. Understand how to create and manipulate database tables using SQL.
4. Recognize key database terms such as tables, primary keys, and foreign keys.
5. Develop logical understanding of database design using SQL concepts.

THEORETICAL BACKGROUND

Introduction to SQL

SQL (Structured Query Language) is a standard programming language designed for managing and interacting with databases. It enables users to store, organize, update, and retrieve data efficiently. SQL serves as a language to communicate with databases. It forms the backbone of all modern database management systems and is supported by major platforms such as MySQL, Oracle, SQL Server and PostgreSQL.

SQL provides a systematic and powerful way to handle large volumes of data, making it an essential skill for anyone working in data management, software development or information systems.

IMPORTANCE OF SQL

1. SQL helps **organize and manage** large amounts of data.
2. It allows multiple users to **access and modify data** at the same time.

3. It ensures data **consistency** and **accuracy**.
4. It is widely used in almost every field such as **banking, education, business and research**.
5. SQL provides **powerful security features** to protect sensitive information.

FUNCTIONS OF SQL

SQL performs several key functions in database management. These functions are grouped into categories based on their purpose:

Function	Description	Example
Data Definition (DDL)	Defines the structure of the database such as tables and columns.	CREATE TABLE Student (...);
Data Manipulation (DML)	Used to insert, update, and delete data in tables.	INSERT, UPDATE, DELETE
Data Query (DQL)	Used to fetch or read data from the database.	SELECT * FROM Student;
Data Control (DCL)	Controls access and permissions in a database.	GRANT, REVOKE
Transaction Control (TCL)	Manages transactions and saves changes.	COMMIT, ROLLBACK

TYPES OF SQL COMMANDS

1. Data Definition Language (DDL)

Used to define and modify the structure of the database (tables, columns, etc.).

Command	Purpose
CREATE	Creates a new database or table.
ALTER	Modifies the structure of an existing table.
DROP	Deletes a table or database.

Example: CREATE TABLE Student (Student_ID INT PRIMARY KEY, Name VARCHAR(50), Age INT);

2. Data Manipulation Language (DML)

Used to manage and change data inside tables.

Command	Purpose
INSERT	Add new records.
UPDATE	Modify existing records.
DELETE	Remove records.

Example: INSERT INTO Student (Student_ID, Name, Age) VALUES (101, 'Ali', 20);

3. Data Query Language (DQL)

Used to retrieve data from database tables.

Example: SELECT Name, Age FROM Student;

4. Data Control Language (DCL)

Used to control access and permissions to the database.

Commands: GRANT (give permission), REVOKE (remove permission).

5. Transaction Control Language (TCL)

Used to manage changes made by DML commands.

Commands: COMMIT (save changes), ROLLBACK (undo changes).

KEY FEATURES OF SQL

- Works across multiple database platforms.
- Provides fast and powerful data manipulation capabilities.
- Supports secure access and transaction control.
- Handles large amounts of data efficiently.

COMMON SQL TERMS

Term	Meaning
Database	A collection of related data organized in tables.
Table	Stores data in rows and columns.
Row (Record)	A single entry or data point in a table.

Column (Field)	A specific type of data, like Name or ID.
Primary Key	A unique identifier for each record in a table.
Foreign Key	A field in one table that links it to another table.

DATABASE DESIGN EXAMPLE

Below is an example of a simple database design using SQL concepts. It includes two tables: Student and Department.

Student Table
Student_ID (PK)
Name
Email
Department_ID (FK)

Department Table
Department_ID (PK)
Department_Name
Location

Relationship:
→ Each **Student belongs to one Department**.
→ **Department_ID** in the **Student table** acts as a Foreign Key.

CONCLUSION

SQL is an essential tool for anyone working with databases. It allows users to define, store and retrieve data efficiently while maintaining data integrity and security. Understanding SQL is a key step in becoming proficient in database design and management.

LAB TASK

Task 1

Scenario:
Design a schema for a Course Enrollment System.

Tables:

- Student (Student_ID PK, Name, Email)
- Course (Course_ID PK, Title, Credit_Hours)
- Enrollment (Enroll_ID PK, Student_ID FK, Course_ID FK, Date)

Task:

- Draw all three tables in draw.io.
- Show connections:
    - Student → Enrollment
    - Course → Enrollment
- Label the cardinalities properly (1–M, M–1).
- Write relationship names (e.g., “Enrolls In”, “Includes”).

Task 2

Scenario:
Design a **Company Database Schema** with the following tables:

Tables:

- **Employee** (Emp_ID PK, Name, Designation, Dept_ID FK)
- **Department** (Dept_ID PK, Dept_Name, Location)
- **Project** (Proj_ID PK, Title, Budget, Dept_ID FK)
- **Client** (Client_ID PK, Client_Name, Contact_Person, Proj_ID FK)

Task:

- Create four connected tables in draw.io.

- Use arrows to show:
    - Department → Employee
    - Department → Project
    - Project → Client
- Clearly mark all PKs and FKs.
- Add note: Each Project belongs to one Department and is assigned to one Client.

Task 3

Scenario:
Build a University Database Schema with the following tables:

- Student (Student_ID PK, Name, Email, DOB, Dept_ID FK)
- Department (Dept_ID PK, Dept_Name, Location)
- Course (Course_ID PK, Title, Credit_Hours, Dept_ID FK)
- Instructor (Inst_ID PK, Name, Email, Dept_ID FK)
- Registration (Reg_ID PK, Student_ID FK, Course_ID FK, Inst_ID FK, Grade)

Task:

Draw all 5 tables in draw.io.
Connect relationships properly using arrows:

- Department → Student
- Department → Course
- Department → Instructor
- Student → Registration
- Course → Registration
- Instructor → Registration

- Label all Primary and Foreign Keys.
- Define cardinalities (1–M, M–N) on the diagram.
- Mention at least 3 SQL CREATE TABLE statements below the diagram as text boxes (just structure, no execution needed).

Task 4

Scenario:

You are required to design a database for an E-Commerce Store that manages customers, products, orders, employees and payments. The system should store all necessary information to track customer orders, product details, employee handling and payment records.

Business Rules:

- Customers can place multiple orders.
- Each order can include multiple products.
- Products belong to specific categories.
- Each order is handled by one employee (salesperson).
- Payments are recorded for each order.

Tables to Include and Their Attributes:

1. **Customer**
    - Customer_ID (PK)
    - Name
    - Email
    - Contact
    - Address
2. **Employee**
    - Emp_ID (PK)
    - Name
    - Email
    - Designation
3. **Category**
    - Category_ID (PK)
    - Category_Name

4. **Product**
    - Product_ID (PK)
    - Product_Name
    - Price
    - Category_ID (FK)
5. **Orders**
    - Order_ID (PK)
    - Order_Date
    - Customer_ID (FK)
    - Emp_ID (FK)
6. **Order_Detail** (Junction Table for M–N Relationship)
    - Order_ID (FK)
    - Product_ID (FK)
    - Quantity
    - Total_Amount
7. **Payment**
    - Payment_ID (PK)
    - Order_ID (FK)
    - Payment_Date
    - Amount
    - Payment_Mode

Tasks

1. Draw the **Complete Schema** in **draw.io**.
2. Represent each table as a separate box with all attributes listed.
3. **Underline** all Primary Keys (PK).
4. **Label** all Foreign Keys (FK).

5. Use **arrows notation** to represent relationships:
    - Customer → Orders
    - Employee → Orders
    - Orders → Order_Detail
    - Product → Order_Detail
    - Category → Product
    - Orders → Payment
6. Clearly define **cardinalities** (1–M, M–N).
7. Ensure all relationships are logically connected and accurately represented.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric

Lab # 3 **Lab Title :** Construct Database Schema based on SQL Modeling **Section :** I

Areas Assessed		Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
		(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)		Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)		Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)		Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr #	Maximum Marks ►	2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼ Student Name ▼											
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												

Implementation of DDL Commands of SQL (Create, Alter, Drop Table)

OBJECTIVES

1. To understand the structure and purpose of **SQL and RDBMS**.
2. To learn how to **define, create and organize** database tables using SQL.
3. To explore and apply **DDL (Data Definition Language)** commands for creating and modifying database structures.
4. To develop the ability to **alter, rename and delete** tables and columns as per system requirements.
5. To gain practical experience in **database schema management** for real-world applications.

OUTCOMES

After completing this experiment, students will be able to:

1. Understand the concept and usage of **DDL commands** in SQL.
2. Create database tables and define appropriate **data types** for each field.
3. Add, modify, rename or delete table columns using **ALTER TABLE**.
4. Rename or drop entire tables using appropriate SQL syntax.
5. Design a simple database schema independently and manage its structure confidently.
6. Demonstrate a clear understanding of **database design and schema control** in practical scenarios.

THEORETICAL BACKGROUND

Introduction to SQL

SQL (Structured Query Language) is a standard language used to manage and interact with data in a **Relational Database Management System (RDBMS)**.
It is used for:

- Creating and modifying database structures.
- Inserting, updating and deleting data.
- Retrieving information from databases.
- Controlling data access and transactions.

MYSQL TOOL

MySQL provides several tools to work with databases:

Tool	Description
MySQL Workbench	A graphical interface to design, create and manage databases. It allows you to write SQL queries, manage tables and visualize database structures.
Command-Line	Allows you to execute SQL commands directly from the terminal
MySQL Shell	Advanced tool for executing SQL and JavaScript commands, suitable for scripting and database automation.
MySQL Reports / Export Tools	Tools within MySQL Workbench to generate reports and export data in different formats (CSV, JSON etc.).
MySQL Visualizations	MySQL Workbench supports simple charts and graphs for visual representation of query results.

TYPES OF SQL COMMANDS

SQL commands are grouped into **five main categories:**

Type	Full Form	Purpose
DDL	Data Definition Language	Define database structure
DML	Data Manipulation Language	Insert, update, delete data
DRL / DQL	Data Retrieval Language	Retrieve data using `SELECT`
TCL	Transaction Control Language	Manage transactions
DCL	Data Control Language	Manage access and permissions

Data Definition Language

DDL is used to **create, modify, and delete** database objects such as tables, schemas, and indexes.

1. CREATE TABLE

Purpose: To create a new table in the database with specified columns and data types.
Description: This command defines the structure of the table including column names, data types, and sizes.
Syntax:

```
CREATE TABLE table_name (
  column1 datatype(size),
  column2 datatype(size),
  ...
);
```

Example:

```
CREATE TABLE Student (
  Sno NUMBER(3),
  Sname CHAR(10),
  Class CHAR(5)
);
```

2. ALTER TABLE

Purpose: To modify an existing table structure without deleting it.
Description: You can add, modify, drop, or rename columns using this command.

(a) Add a New Column

Adds a new column to an existing table.

ALTER TABLE Student ADD (Address CHAR(20));
Description: Adds a column Address to the Student table.

(b) Modify a Column (Change Type or Size)

Changes the data type or size of an existing column.

ALTER TABLE Student MODIFY (Sname VARCHAR2(15));
Description: Changes Sname column to allow up to 15 characters.

(c) Drop a Column

Removes a column from the table permanently.

ALTER TABLE Student DROP COLUMN Address;
Description: Deletes the Address column from the Student table.

(d) Rename a Column

Changes the name of an existing column.

ALTER TABLE Student RENAME COLUMN Sname TO Stu_Name;
Description: Renames Sname column to Stu_Name.

3. DROP TABLE

Purpose: To delete a table completely from the database, along with all its data.
Description: Use this command carefully because all records will be permanently lost.

DROP TABLE Student;
Deletes the Student table entirely.

4. RENAME TABLE

Purpose: To change the name of an existing table.
Description: This is useful for organizing tables or renaming for better clarity.

RENAME Student TO Student_Data;
Changes the table name from Student to Student_Data.

PURPOSE OF DDL

- To **define and create the structure** of database objects such as tables, schemas, and indexes.
- To **modify existing database structures** without affecting the data (e.g., adding or renaming columns).
- To **remove or rename** database objects when they are no longer needed.
- To provide a **standardized way** to manage database schema across different database systems.

IMPORTANCE OF DDL

- **Foundation of Database Design:** DDL commands allow you to create the framework on which all data is stored.
- **Data Integrity:** By defining correct data types and constraints, DDL ensures that the stored data is accurate and consistent.
- **Flexibility:** DDL allows you to update the database structure as requirements change without losing existing data.
- **Control and Management:** Helps database administrators manage schema, maintain organization, and optimize storage.
- **Standardization:** DDL commands are standardized in SQL, making it easier to work across different RDBMS like MySQL, Oracle, and SQL Server.

LAB TASK

1. Create a table EMPLOYEE with following schema:
   (Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name, Job_id , Salary)
   Hint: Use the `CREATE TABLE` command and define suitable data types.

2. Add a new column; HIREDATE to the existing relation.
   Hint: Use `ALTER TABLE` with `ADD` option.

3. Change the datatype of JOB_ID from char to varchar2.
   Hint: Use `ALTER TABLE` with `MODIFY` option.

4. Change the name of column/field Emp_no to E_no.
   Hint: Use `ALTER TABLE` with `RENAME COLUMN` option.

5. Modify the column width of the job field of emp table
   Hint: Use `ALTER TABLE` with `MODIFY` option to adjust size.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric

Lab # 4 **Lab Title :** Implementation of DDL commands of SQL **Section :** I

Areas Assessed			Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)			Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)			Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)			Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr #	Maximum Marks ►		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Implementation of DQL & DML Commands of SQL (SELECT, FROM & WHERE Commands)(INSERT, UPDATE & DELETE Tables)

OBJECTIVES

1. To understand the purpose and usage of **Data Manipulation Language** (DML) in SQL.
2. To learn how to **add, modify, and delete** records in database tables.
3. To practically implement SQL **INSERT, UPDATE, DELETE, and SELECT** commands.
4. To apply conditional data manipulation using the **WHERE clause**.
5. To build confidence in handling real-world database operations.

OUTCOMES

After completing this experiment, students will be able to:

1. Understand DML operations in relational databases.
2. Insert new records using the INSERT INTO command.
3. Update existing data using the UPDATE command with conditions.
4. Delete specific or all rows using the DELETE command.
5. Retrieve data using SELECT for verification.
6. Apply SQL syntax confidently in MySQL or any online SQL compiler.

PRE-REQUISITES

Students should already know:

- Basics of database tables and data types.
- How to create tables using SQL DDL (CREATE, ALTER).
- Understanding of columns, rows, and constraints.

THEORETICAL BACKGROUND

SQL (Structured Query Language) is used to work with data stored in a **Relational Database Management System (RDBMS)**.

SQL commands are divided into categories:

Category	Commands	Purpose
DDL	CREATE, ALTER, DROP	Structure of tables
DML	INSERT, UPDATE, DELETE	Manipulate table data
DQL	SELECT	Retrieve data

DML commands **do not change table structure** they only change the **data inside the table**.

SQL SELECT STATEMENT (RETRIEVING DATA)

The **SELECT** command is used to show the data stored in the table.
You can retrieve:
1. One column
2. Multiple columns
3. All columns
4. Filtered records using WHERE

5.1 SELECT Syntax

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

Here, column1, column2, ... are the field names of the table you want to select data from.

To retrieve **all columns**:

```
SELECT * FROM table_name;
```

* Means Select All Columns

5.2 Why SELECT Comes First?

Because without retrieving data, the user cannot see what's inside the table. It is the **most commonly used SQL command** and helps verify if INSERT, UPDATE, DELETE worked correctly.

EAXAMPLE DATABASE TABLE

SQL commands are grouped into **five main categories:**

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedadosy Helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA11DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL QUERY FROM ABOVE TABLE

1. SQL SELECT & FROM QUERY

i. Selecting Specific Columns From above table

Query:

```
SELECT Customer Name, City
FROM Customers;
```

Explanation:

This retrieves **only 2 columns**, not the entire table. Useful when you need summary data.

ii. Selecting All Columns

Query:

```
SELECT * FROM Customers;
```

Explanation:

Shows **complete table** with all fields.

2. SQL WHERE Clause (Filtering Records)

The WHERE clause is used to retrieve **only specific rows**.

Syntax:

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

i. Text Field Condition

```
SELECT * FROM Customers
WHERE Country = 'Mexico';
```

Text values always use **single quotes**.

ii. Numeric Field Condition

```
SELECT * FROM Customers
WHERE CustomerID = 1;
```

Numbers do **not** use quotes.

iii. Operators Used in WHERE

Operator	Meaning
=	Equal to
>	Greater than
<	Less than
>=	Greater or equal
<=	Less or equal
<>	Not equal

BETWEEN	Range check
LIKE	Pattern matching
IN	Multiple values

SQL INSERT STATEMENT (ADDING DATA)

The **INSERT** command is used to add new rows into a table.
INSERT is placed **before UPDATE and DELETE** because:

1. Data must exist before you can update or delete it.
2. INSERT builds the initial dataset.

1. INSERT Syntax (All Columns)

```
INSERT INTO table_name
VALUES (value1, value2, value3);
```

Use this when inserting values **in the same order as table structure**.

2. INSERT Syntax (Specific Columns)

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

Use this when inserting into only a few columns.

Example — Insert a Customer

```
INSERT INTO Customers (Customer ID, Customer Name, City, Country)
VALUES (6, 'Hamza Traders', 'Lahore', 'Pakistan');
```

Explanation:

- A new row is added
- The order of columns matches the order of values
- This command does not affect existing data

SQL UPDATE STATEMENT (MODIFYING DATA)

UPDATE is used to change an existing value in the table.

Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2
```

```
WHERE condition;
```

The WHERE clause is important without it, **entire table will be changed**.

I. **Example — Update One Record**

```
UPDATE Customers
SET Contact Name = 'Alfred Schmidt', City = 'Frankfurt'
WHERE Customer ID = 1;
```

Explanation:

- Only record with `Customer ID = 1` is updated
- Useful for correcting wrong information

II. **Example — Update Multiple Records**

```
UPDATE Customers
SET ContactName = 'Juan'
WHERE Country = 'Mexico';
```

Explanation:

All customers in Mexico will have ContactName changed to “Juan”

III. **Example — Dangerous Update (NO WHERE)**

```
UPDATE Customers
SET ContactName = 'Juan';
```

Explanation:

Every row is updated — not recommended

SQL DELETE STATEMENT (REMOVE DATA)

DELETE removes data from table rows.

Syntax:

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

I. **Example — Delete a Specific Record**

```
DELETE FROM Customers
WHERE Customer Name = 'Alfreds Futterkiste';
```

Explanation:

- Only one row is deleted
- Table structure does not change

II. **Delete All Records**

```
DELETE FROM Customers;
```

Explanation:

Removes all rows but **keeps table structure**

LAB TASK

EmpNo	EmpName	Designation	OfficeNo	Hiredate	Sales	Commission	DepNo
7369	SALMAN	CLERK	7902	17-Dec-00	8000		20
7499	ALIA	SALESMAN	7698	20-Feb-01	16000	3000	30
7521	WAHEED	SALESMAN	7698	22-Feb-01	12500	5000	30
7566	JAMIL	MANAGER	7839	2-Apr-01	29750		20
7654	MEHVISH	SALESMAN	7698	28-Sep-01	12000	12500	30
7698	BILAL	MANAGER	7839	1-May-01	28500		30
7782	QASIM	MANAGER	7839	9-Jun-01	24500		10
7788	SHAMS	ANALYST	7566	9-Dec-02	30000		20
7839	KAMRAN	PRESIDENT		17-Nov-01	50000		10
7844	TANVEER	SALESMAN	7698	8-Sep-01	15000	0	30
7876	AFTAB	CLERK	7788	12-Jan-03	11000		20
7900	JAVERIA	CLERK	7698	3-Dec-01	9500		30
7902	FOZIA	ANALYST	7566	3-Dec-01	30000		20
7934	MURAD	CLERK	7782	23-Jan-02	13000		10

Task 1:

Create the **Employee** table using the structure shown above and insert **all the given records** using appropriate INSERT INTO statements.

Task 2:

Display the **EmpName** and **Commission** of all employees.

Task 3:

Display the **EmpNo**, **EmpName**, and **Designation** of all employees who work in **Department 30**.

Task 4:

Display the **EmpName**, **EmpNo**, and **OfficeNo** of all employees whose **Designation is 'CLERK'**.

Task 5:

Display the details of all employees whose **Commission is greater than Sales**.

Task 6:

Display all employees whose **OfficeNo = 7782** (i.e., employees managed by office/manager 7782).

Task 7:

Insert the following **new employee record** into the table:
(8001, 'Hassan', 'Analyst', 5466, '26-Oct-2019', 500000, 0, 20)

Task 8:

Update the record of the employee with **EmpNo = 7839** and set their **OfficeNo = 7777**.

Task 9:

Delete the record of the employee whose **EmpName = 'WAHEED'**.

NOTE FOR STUDENTS:

- Each group must write their own SQL queries. Copying from others or online is not allowed.
- All queries should be attached with manual.
- Print the final database table and attach them to your lab manual.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric													
Lab # 5 Lab Title : Implementation of DML & DQL commands of SQL Section : I													
Areas Assessed			Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)			Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)			Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)			Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr #	Maximum Marks ►		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Implementation of different types of function of SQL (Number, Aggregate, Character, Conversion & Date Functions)

OBJECTIVES

1. To understand the purpose and use of different SQL functions.
2. To learn how number, aggregate, character, conversion, and date functions work.
3. To practice SQL queries using built-in functions.
4. To apply functions on data to perform calculations and formatting.
5. To build confidence in retrieving and processing data using SQL functions.

OUTCOMES

After completing this experiment, students will be able to:

1. Use number functions for mathematical calculations.
2. Apply aggregate functions for summarizing data.
3. Use character functions to modify and format text.
4. Convert data types using conversion functions.
5. Handle and manipulate date values using date functions.
6. Write SQL queries confidently using function-based operations.

PRE-REQUISITES

Students should already know:

- Basics of SQL and relational databases.
- How to create and insert records in tables.
- Basic SQL commands like SELECT and WHERE.
- Understanding of data types (number, text, date).

THEORETICAL BACKGROUND

SQL provides different types of **built-in functions** to perform operations on data.
Functions help in calculating values, summarizing information, formatting text, converting data types, and manipulating date values.

SQL Functions are mainly classified as:

Category	Description
Number Functions	Perform mathematical calculations
Aggregate Functions	Summarize multiple rows (SUM, MAX, MIN, etc.)
Character Functions	Modify or format text values
Conversion Functions	Convert data types (TO_CHAR, TO_NUMBER)
Date Functions	Perform operations on dates

These functions make SQL more powerful by allowing us to process data efficiently.

1. NUMBER FUNCTION

Number functions perform mathematical calculations on numeric values.

Function	Purpose	Example
ABS(n)	Returns absolute value	SELECT ABS(-15) FROM dual;
EXP(n)	Raises e to the power n	SELECT EXP(4) FROM dual;
POWER(m,n)	m raised to n	SELECT POWER(4,2) FROM dual;
MOD(m,n)	Remainder of division	SELECT MOD(10,3) FROM dual;
ROUND(m,n)	Rounds number n decimal places	SELECT ROUND(100.256,2) FROM dual;
TRUNC(m,n)	Truncates number without rounding	SELECT TRUNC(100.256,2) FROM dual;
SQRT(n)	Returns square root	SELECT SQRT(16) FROM dual;

2. AGGEREGATED FUNCTION

Function	Purpose	Example
COUNT(column)	Returns the number of rows	SELECT COUNT(Sal) FROM emp;
SUM(column)	Adds all values in a column	SELECT SUM(Sal) FROM emp;
AVG(column)	Returns the average value	SELECT AVG(10 + 15 + 30) FROM dual;
MAX(column)	Returns the highest value	SELECT MAX(Sal) FROM emp;
MIN(column)	Returns the lowest value	SELECT MIN(Sal) FROM emp;

3. CHARACTER FUNCTION

Character functions help in modifying text or strings.

Function	Purpose	Example
INITCAP	Converts first letter to uppercase	SELECT INITCAP('hello') FROM dual;
LOWER	Converts to lowercase	SELECT LOWER('HELLO') FROM dual;
UPPER	Converts to uppercase	SELECT UPPER('hello') FROM dual;
LTRIM	Removes characters from left side	SELECT LTRIM('cseit','cse') FROM dual;
RTRIM	Removes characters from right side	SELECT RTRIM('cseit','it') FROM dual;
REPLACE	Replaces characters	SELECT REPLACE('jack and jue','j','bl') FROM dual;

4. CONVERSION FUNCTION

Function	Purpose	Example
TO_CHAR (value, format)	Converts a number or date into text format	SELECT TO_CHAR(65,'RN') FROM dual;
TO_NUMBER (text)	Converts a text value into a number	SELECT TO_NUMBER('1234.64') FROM dual;
TO_DATE (text)	Converts text into a date format	SELECT TO_DATE('January 15, 1989') FROM dual;

5. DATE FUNCTION

SQL provides date functions to work with date values.

Function	Purpose	Example
SYSDATE	Shows current system date	SELECT SYSDATE FROM dual;
NEXT_DAY	Next specific weekday	SELECT NEXT_DAY(SYSDATE,'WED') FROM dual;
ADD_MONTHS	Adds months	SELECT ADD_MONTHS(SYSDATE,2) FROM dual;
LAST_DAY	Last day of month	SELECT LAST_DAY(SYSDATE) FROM dual;
MONTHS_BETWEEN	Difference in months	SELECT MONTHS_BETWEEN(SYSDATE,HIREDATE) FROM emp;
LEAST	Earliest date	SELECT LEAST('10-JAN-07','12-OCT-07') FROM dual;
GREATEST	Latest date	SELECT GREATEST('10-JAN-07','12-OCT-07') FROM dual;
TRUNC	Removes time part	SELECT TRUNC(SYSDATE,'DAY') FROM dual;
ROUND	Rounds date	SELECT ROUND(SYSDATE,'DAY') FROM dual;

EXAMPLE OF COMPANY MANAGEMENT SYSTEM

Employee:

EmpID	EmpName	Job	Salary	Commission	HireDate	DeptID	ManagerID
1001	Ali Raza	Manager	80000	0	10-JAN-2018	10	NULL
1002	Sara Khan	Analyst	60000	5000	12-MAR-2019	20	1001
1003	Ahmed Noor	Clerk	30000	0	05-MAY-2020	30	1002
1004	Fatima Ali	Salesman	45000	10000	15-JUL-2017	20	1001
1005	Hamza Iqbal	Analyst	55000	2000	08-AUG-2021	10	1001
1006	Zain Malik	Salesman	47000	15000	20-OCT-2018	30	1002
1007	Sameer Khan	Clerk	25000	0	11-DEC-2022	40	1003
1008	Ayesha Noor	Manager	90000	0	04-MAR-2016	40	NULL
1009	Waheed	Salesman	50000	8000	19-JUN-2019	20	1004
1010	Mariam Ali	Analyst	62000	0	23-FEB-2023	10	1001

1. NUMBER FUNCTIONS

- **ABS() – Absolute difference between Salary & Commission**

```
SELECT EmpName, ABS(Salary - Commission) AS Difference
FROM Employee;
```

- **MOD() – Check if Salary is even or odd**

```
SELECT EmpID, EmpName, MOD(Salary, 2) AS SalaryParity
FROM Employee;
```

- **POWER() – Square of Salary in thousands**

```
SELECT EmpName, POWER(Salary/1000, 2) AS SalaryPower
FROM Employee;
```

- **ROUND() – Monthly salary rounded to 2 decimals**

```
SELECT EmpName, ROUND(Salary/12, 2) AS MonthlySalary
FROM Employee;
```

- **SQRT() – Square root of Salary**

```
SELECT EmpName, SQRT(Salary) AS SalaryRoot
FROM Employee;
```

2. AGGREGATE FUNCTIONS

- **COUNT() – Total employees**

```
SELECT COUNT(*) AS TotalEmployees
FROM Employee;
```

- **SUM() – Total salary per department**

```
SELECT DeptID, SUM(Salary) AS TotalDeptSalary
FROM Employee
```

- **AVG() – Average salary of all employees**

```
SELECT AVG(Salary) AS AverageSalary
FROM Employee;
```

- **MAX() – Highest salary**

```
SELECT MAX(Salary) AS HighestSalary
FROM Employee;
```

- **MIN() – Lowest salary**

```
SELECT MIN(Salary) AS LowestSalary
FROM Employee;
```

3. CHARACTER FUNCTIONS

- **LOWER() – Employee name in lowercase**

```
SELECT LOWER(EmpName) AS LowerName
```

```
FROM Employee;
```

- **UPPER() – Job title in uppercase**

```
SELECT UPPER(Job) AS UpperJob
FROM Employee;
```

- **INITCAP() – Capitalize first letters**

```
SELECT INITCAP(EmpName) AS InitName
FROM Employee;
```

- **REPLACE() – Replace "Clerk" with "Assistant"**

```
SELECT EmpName, REPLACE(Job,'Clerk','Assistant') AS UpdatedJob
FROM Employee;
```

- **LTRIM() / RTRIM() – Trim unwanted characters**

```
SELECT LTRIM('___Ali Raza','_') AS TrimmedName FROM dual;
SELECT RTRIM('Fatima Ali___','_') AS TrimmedName FROM dual;
```

4□. CONVERSION FUNCTIONS

- **TO_CHAR() – Format HireDate**

```
SELECT EmpName, TO_CHAR(HireDate, 'DD-MON-YYYY') AS JoiningDate
FROM Employee;
```

- **TO_NUMBER() – Add a bonus of 50000**

```
SELECT EmpName, Salary + TO_NUMBER('50000') AS SalaryWithBonus
FROM Employee;
```

- **TO_DATE() – Convert string to date**

```
SELECT EmpName, TO_DATE('01-01-2024','DD-MM-YYYY') AS NewYear
FROM Employee;
```

5. DATE FUNCTIONS

- **SYSDATE – Current system date**

```
SELECT EmpName, SYSDATE AS TodayDate
FROM Employee;
```

- **ADD_MONTHS() – Date after 6 months from HireDate**

```
SELECT EmpName, ADD_MONTHS(HireDate, 6) AS IncrementDate
FROM Employee;
```

- **LAST_DAY() – Last day of current month**

```
SELECT LAST_DAY(SYSDATE) AS MonthEnd FROM dual;
```

- **NEXT_DAY() – Next Monday from today**

```
SELECT NEXT_DAY(SYSDATE, 'MONDAY') AS NextMonday FROM dual;
```

- **MONTHS_BETWEEN() – Months worked since HireDate**

```
SELECT EmpName, MONTHS_BETWEEN(SYSDATE, HireDate) AS MonthsWorked
FROM Employee;
```

LAB TASK

Create a table EMPLOYEE with following schema: (Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name,Job_id, Designation , Salary, Hire_date)

Emp_no	E_name	E_address	E_ph_no	Dept_no	Dept_name	Job_id	Designation	Salary	Hire_date
1001	Ali Raza	Karachi	3001234567	10	HR	1	MANAGER	80000	10-Jan-2018
1002	Sara Khan	Lahore	3007654321	20	IT	2	ANALYST	60000	12-Mar-2019
1003	Ahmed Noor	Islamabad	3009876543	10	HR	3	CLERK	30000	05-May-1981
1004	Hina Malik	Karachi	3003456789	20	IT	4	IT PROFF	50000	17-Dec-1981
1005	Samiya Tariq	Lahore	3007659876	30	FINANCE	5	CLERK	32000	01-May-1981
1006	Haris Ali	Islamabad	3002345678	10	HR	6	ANALYST	45000	19-Jan-1980
1007	Shahid Khan	Lahore	3005678901	20	IT	7	MANAGER	75000	03-Dec-1981
1008	Salman Ahmed	Karachi	3004567890	10	HR	8	PRESIDENT	90000	15-Jan-1979
1009	Hafsa Qureshi	Lahore	3006789012	30	FINANCE	9	CLERK	31000	17-Dec-1981
1010	Hassan Raza	Islamabad	3007890123	20	IT	10	ANALYST	47000	12-May-1982
1011	Sana Iqbal	Karachi	3003451234	10	HR	11	CLERK	34000	01-May-1981
1012	Hamza Siddiqui	Lahore	3004561234	20	IT	12	MANAGER	77000	10-Jun-1983

Task 1:

List the E_no, E_name, Salary of all employees working for MANAGER.

Task 2:

Display all the details of the employee whose salary is more than the Salary of any IT PROFF

Task 3:

List the employees in the ascending order of Designations of those joined after 1981.

Task 4:

List the employees who are either ‘CLERK’ or ‘ANALYST’ .

Task 5:

List the employees who joined on 1-MAY-81, 3-DEC-81, 17-DEC-81,19-JAN-80.

Task 6:

List the employees who are working for the Deptno 10 or20.

Task 7:

List the Enames those are starting with ‘S’ .

Task 8:

Dislay the name as well as the first five characters of name(s) starting with ‘H’.

Task 9:

List all the emps except ‘PRESIDENT’ & ‘MGR” in asc order of Salaries.

NOTE FOR STUDENTS:

- Each group must write their own SQL queries. Copying from others or online is not allowed.
- Print all tasks and attach them to your lab manual.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric

Lab # 6 **Lab Title :** Implementation of different types of function **Section :** I

Areas Assessed			Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)			Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)			Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)			Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr #	Maximum Marks ▶		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Implementation of different types of Operators of SQL

OBJECTIVES

1. Understand what operators are and why they are used in SQL.
2. Learn the purpose and use of different SQL operators.
3. Write SQL queries using arithmetic, logical, comparison, special and set operators.
4. Apply operators to filter, compare, and retrieve data from tables.

THEORETICAL BACKGROUND

Operators in SQL are symbols or keywords used to perform calculations, compare values or apply conditions in a query.
They help us manipulate data and make decisions while writing SQL statements.

SQL operators are mainly classified into:

1. **Arithmetic Operators**
2. **Logical Operators**
3. **Comparison Operators**
4. **Special Operators**
5. **Set Operators**

1. ARITHMETIC OPERATORS

Operator	Meaning	Example Explanation
+	Addition	Adds two values
-	Subtraction	Subtracts right value from left value
*	Multiplication	Multiplies values
/	Division	Divides left value by right value
^	Power	Raises a number to the power of another number
%	Modulus	Returns remainder after division

Examples

```
SELECT 10 + 5 AS Addition;
SELECT 20 - 7 AS Subtraction;
SELECT 6 * 3 AS Multiplication;
SELECT 10 / 2 AS Division;
SELECT 2 ^ 3 AS PowerResult;
SELECT 10 % 3 AS Remainder;
```

2. LOGICAL OPERATORS

Logical operators are used in the **WHERE** clause to combine conditions.

Operator	Purpose
AND	Returns TRUE only if **all conditions** are true
OR	Returns TRUE if **any one condition** is true
NOT	Reverses the result of a condition (TRUE becomes FALSE)

Examples

No1:

```
SELECT * FROM Employee
WHERE Salary > 50000 AND Job = 'Analyst';
```

No2:

```
SELECT * FROM Employee
WHERE DeptID = 10 OR DeptID = 20;
```

No3:

```
SELECT * FROM Employee
WHERE NOT Job = 'Clerk';
```

3. COMPARISON OPERATORS

These operators compare two values and return TRUE or FALSE.

Operator	Meaning
=	Equal to
!= or <>	Not equal to
>	Greater than

Operator	Meaning
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Examples

```
SELECT * FROM Employee
WHERE Salary = 60000;

SELECT * FROM Employee
WHERE Salary <> 30000;

SELECT * FROM Employee
WHERE Salary > 50000;

SELECT * FROM Employee
WHERE Salary <= 45000;
```

4. SPECIAL OPERATORS

Special operators help search for specific patterns, ranges, or check conditions.

- BETWEEN

  Checks if a value is between two given values (inclusive).

  ```
  SELECT * FROM Employee
  WHERE Salary BETWEEN 30000 AND 60000;
  ```

- IS NULL

  Checks if a column contains NULL value.

  ```
  SELECT * FROM Employee
  WHERE Commission IS NULL;
  ```

- LIKE

  Used to match text patterns using wildcards:

- % = any number of characters
- _ = one character

```
SELECT * FROM Employee
WHERE EmpName LIKE 'A%';
```

- IN

  Used to match any value from a list.

```
SELECT * FROM Employee
WHERE DeptID IN (10, 20, 30);
```

- ANY

  Returns TRUE if **any one value** in the list matches the condition.

```
SELECT * FROM Employee
WHERE Salary > ANY (SELECT Salary FROM Employee WHERE DeptID = 10);
```

- ALL

  Returns TRUE only if **all values** in the list match the condition.

```
SELECT * FROM Employee
WHERE Salary > ALL (SELECT Salary FROM Employee WHERE DeptID = 20);
```

5. SET OPERATORS

Set operators combine the results of two SELECT queries.

Operator	Purpose
UNION	Combines results of both queries and removes duplicates
UNION ALL	Combines all results including duplicates
INTERSECT	Returns only rows that appear in both queries
MINUS	Returns rows from the first query that are NOT in the second query

Examples

```
SELECT EmpName FROM Employee WHERE DeptID = 10
UNION
SELECT EmpName FROM Employee WHERE DeptID = 20;

SELECT EmpName FROM Employee WHERE Job = 'Analyst'
INTERSECT
SELECT EmpName FROM Employee WHERE Salary > 50000;

SELECT EmpID FROM Employee
MINUS
SELECT ManagerID FROM Employee;
```

LAB TASK

Create a table EMPLOYEE with following schema:

EmpNo	EmpName	Designation	OfficeNo	Sales	Commission	DepNo	City
7369	SALMAN	CLERK	7902	8000	0	20	Lahore
7499	ALIA	SALESMAN	7698	16000	3000	30	Karachi
7521	WAHEED	SALESMAN	7698	12500	5000	30	Lahore
7566	JAMIL	MANAGER	7839	29750	0	20	Islamabad
7654	MEHVISH	SALESMAN	7698	12000	12500	30	Karachi
7698	BILAL	MANAGER	7839	28500	0	30	Lahore
7782	QASIM	MANAGER	7839	24500	0	10	Islamabad
7788	SHAMS	ANALYST	7566	30000	0	20	Lahore
7839	KAMRAN	PRESIDENT	7788	50000	0	10	Karachi
7844	TANVEER	SALESMAN	7698	15000	0	30	Lahore
7876	AFTAB	CLERK	7788	11000	0	20	Islamabad
7900	JAVERIA	CLERK	7698	9500	0	30	Lahore
7902	FOZIA	ANALYST	7566	30000	0	20	Karachi
7934	MURAD	CLERK	7782	13000	0	10	Islamabad

Task 1:

Display **EmpName**, (**Current) Sales**, and **(Updated) Sales** after increasing Sales by 10%.

Task 2:

Display **Employee Name**, **Sales**, and **(Average Daily) Sales** (assume 30 days in a month).

Task 3:

Write a query to list all employees who are **Salesman** AND belong to **Department 30**.

Task 4:

Write a query to display all employees whose designation is either **Manager OR Analyst**.

Task 5:

Write a query to display the employees whose **Sales are greater than 20,000**.

Task 6:

Write a query to display the employees whose **Commission is equal to 0**.

Task 7:

Write a query to display the employees whose **Sales are between 10,000 and 20,000**.

Task 8:

Write a query to list all employees whose **name starts with the letter 'S'**.

Task 9:

Write a query to display the employees who belong to either **Lahore or Karachi**.

Task 10:

Write a query using **UNION** to display all employees who are either **Salesman or Clerk**, without showing duplicates.

NOTE FOR STUDENTS:

- Each group must write their own SQL queries. Copying from others or online is not allowed.
- Print all tasks and attach them to your lab manual.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric													
Lab # 7 Lab Title : Implementation of different types of operators in SQL Section : I													
Areas Assessed		Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total					Total
		(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)						
Poor (0 to 30%)		Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4		
Satisfactory (31 to 70%)		Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork						
Very Good (71 to 100%)		Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work						
Sr #	Maximum Marks ▶		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Implementation of MYSQL Constraints And Relationships

OBJECTIVES

1. Understand the concept of SQL constraints.
2. Learn how to apply different constraints while creating and modifying tables.
3. Understand how constraints help maintain data accuracy, reliability and integrity.

THEORETICAL BACKGROUND

MySQL constraints and relationships are rules that the database uses to keep your data accurate and meaningful. When you store information in tables, you must make sure the data is valid, consistent, and correctly linked. Constraints help enforce these rules automatically.

WHY CONSTRAINTS ARE IMPORTANT?

- They **prevent wrong data** from being entered.
- They **ensure required fields are always filled**.
- They **avoid duplicate values**.
- They **restrict invalid age, marks, salary etc.** using conditions.
- They **assign default values** to save time.

WHAT ARE RELATIONSHIPS?

Relationships connect two tables with each other. For example:

- A **Person** can have many **Orders**.
- A **Student** belongs to one **Department**.
- A **Customer** can have multiple **Invoices**.

This link is created using **FOREIGN KEY**, which ensures:

- You cannot insert an order for a person who does not exist.
- You cannot delete a parent record if child records are dependent.

In short, constraints and relationships make your database **strong**, **organized**, and **error-free**.

SQL CONSTRAINTS OVERVIEW

SQL constraints are rules applied to table columns to ensure valid and meaningful data is stored.

You can apply constraints:

- **When creating a table** → using CREATE TABLE
- **After creating a table** → using ALTER TABLE

General syntax:

```
CREATE TABLE table_name (
column1 datatype constraint,
column2 datatype constraint,
column3 datatype constraint
);
```

1. Not Null Constraint

What it does:

Prevents a column from storing NULL (empty) values. This means the user **must** enter a value.

When to use:

When the data is required (e.g., ID, Name, Email, etc.).

Example (using CREATE TABLE):

```
CREATE TABLE Person (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255) NOT NULL,
   Age int
);
```

Column Name	Data Type	NOT NULL	Accepts NULL	Description
ID	INT	YES	NO	Must always have a value
LastName	VARCHAR(255)	YES	NO	Required
FirstName	VARCHAR(255)	YES	NO	Required
Age	INT	NO	YES	Optional

Why NOT NULL Should Not Be Missing

- If a column is important (like ID, Name, Roll No) it **must always have a value**.
- Without NOT NULL, anyone could insert incomplete data.
- Example: A student without a roll number or an employee without a name.
- Therefore, NOT NULL is required wherever the data should never be empty.

2. Primary Key Constraint

A **Primary Key** is a constraint that uniquely identifies each record (row) in a table.

It has TWO built-in rules:

1. **NOT NULL** — Primary key column cannot be empty.
2. **UNIQUE** — No two rows can have the same value.

Primary Key **always** = NOT NULL + UNIQUE

Example:

```
CREATE TABLE Students (
   StudentID INT,
   Name VARCHAR(100),
   PRIMARY KEY (StudentID)
);
```

Why is Primary Key important?

- It uniquely identifies every row
- It prevents duplicates
- It keeps data organized
- It is used in relationships (FOREIGN KEY depends on it)
- Helps MySQL find records fast (improves performance)

3. Unique Constraint

What it does:

Ensures all values in a column are **different**.

- A table may have **multiple UNIQUE** constraints.
- A PRIMARY KEY automatically includes UNIQUE + NOT NULL.

Example:

```
CREATE TABLE Person (
ID int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255) NOT NULL,
Age int
);
```

Column Name	Data Type	NOT NULL	UNIQUE	Description
ID	INT	YES	YES	No duplicate ID allowed
LastName	VARCHAR(255)	YES	NO	Required
FirstName	VARCHAR(255)	YES	NO	Required
Age	INT	NO	NO	Optional

Why UNIQUE Constraint Is Needed

- UNIQUE makes sure all values are different.
- Perfect for Email, CNIC, Username, or Phone number.
- Prevents duplicate entries that cause confusion.

4. Check Constraint

What it does:

Limits the allowed values in a column.

Example (on CREATE TABLE):

```
Allows only Age ≥ 18:
CREATE TABLE Person (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255) NOT NULL,
Age int CHECK (Age >= 18)
);
```

Column Name	Data Type	NOT NULL	CHECK Rule	Description
ID	INT	YES	—	Required
LastName	VARCHAR(255)	YES	—	Required
FirstName	VARCHAR(255)	YES	—	Optional
Age	INT	NO	Age >= 18 only	Cannot insert age below 18

Add CHECK using ALTER TABLE:

```
ALTER TABLE Person
ADD CHECK (Age >= 18);
```

Explanation: Adds same rule to existing table, no effect on structure except applying rule

Add a named CHECK constraint:

```
ALTER TABLE Person
ADD CONSTRAINT CHK_PersonAge
CHECK (Age >= 18 AND City = 'Sandnes');
```

Column	Condition
Age	Must be ≥ 18
City	Must be 'Sandnes'

Drop CHECK constraint:

```
ALTER TABLE Person
DROP CONSTRAINT CHK_PersonAge;
```

Explanation: Removes the CHECK rule, Table structure stays the same

Why CHECK Constraint Is Needed

- CHECK stops invalid or illogical values.
- Example: Age < 0, Salary < 0, Marks > 100.
- It ensures meaningful and correct data.

5. Default Constraint

What it does:

Automatically assigns a value if no value is provided.

Example (on CREATE TABLE):

```
CREATE TABLE Person (
```

```
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255) NOT NULL,
Age int,
City varchar(255) DEFAULT 'Sandnes'
);
```

Column Name	Data Type	NOT NULL	DEFAULT Value	Description
ID	INT	YES	—	Required
LastName	VARCHAR(255)	YES	—	Required
FirstName	VARCHAR(255)	YES	—	Optional
Age	INT	NO	—	Optional
City	VARCHAR(255)	NO	'Sandnes'	Auto-filled

Drop DEFAULT constraint:

```
ALTER TABLE Person
ALTER COLUMN City DROP DEFAULT;
```

Explanation: Removes automatic default

Why DEFAULT Constraint Helps

- Automatically fills data when the user leaves a field empty.
- Saves time and ensures consistency.
- Example: Default City, Default Account Status ('Active').

6. Foreign Key Constraint

What it does:

Creates a relationship between two tables.

- Refers to a **PRIMARY KEY** of another table.
- The table containing the foreign key is called the **child table**.
- The table being referenced is called the **parent table**.

Example:

```
CREATE TABLE Person (
PersonID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255) NOT NULL ,
```

```
Age int,
City varchar(255) DEFAULT 'Sandnes'
);

CREATE TABLE Orders (
OrderID int PRIMARY KEY,
OrderNumber int NOT NULL,
PersonID int,
FOREIGN KEY (PersonID) REFERENCES Person(PersonID)
);
```

Person Table:

Column	Type	PK	FK	Default
PersonID	INT	YES	No	—
LastName	VARCHAR	No	No	—
FirstName	VARCHAR	No	No	—
Age	INT	No	No	—
City	VARCHAR	No	No	Sandnes

Orders Table:

Column	Type	PK	FK	References
OrderID	INT	YES	No	—
OrderNumber	INT	No	No	—
PersonID	INT	No	YES	Person(PersonID)

Why FOREIGN KEY Is Very Important

- Connects one table with another.
- Protects data using **referential integrity**.
- Prevents inserting child records without a parent.
- Prevents deleting parent records that have linked child data.
- Example: You cannot insert an Order for a Person that does not exist.

Why Ignoring FOREIGN KEY Causes Problems

- Leads to unlinked, broken or invalid records.
- Example:
    - Orders that belong to no customers.

  - Students placed in non-existing departments.
  - Payments stored without matching invoices.
- With foreign keys, data becomes clean, organized, and professionally structured.

LAB TASK

Create a table EMPLOYEE with following schema:

EmpNo	EmpName	Designation	OfficeNo	Sales	Commission	DepNo	City
7369	SALMAN	CLERK	7902	8000	0	20	Lahore
7499	ALIA	SALESMAN	7698	16000	3000	30	Karachi
7521	WAHEED	SALESMAN	7698	12500	5000	30	Lahore
7566	JAMIL	MANAGER	7839	29750	0	20	Islamabad
7654	MEHVISH	SALESMAN	7698	12000	12500	30	Karachi
7698	BILAL	MANAGER	7839	28500	0	30	Lahore
7782	QASIM	MANAGER	7839	24500	0	10	Islamabad
7788	SHAMS	ANALYST	7566	30000	0	20	Lahore
7839	KAMRAN	PRESIDENT	7788	50000	0	10	Karachi
7844	TANVEER	SALESMAN	7698	15000	0	30	Lahore
7876	AFTAB	CLERK	7788	11000	0	20	Islamabad
7900	JAVERIA	CLERK	7698	9500	0	30	Lahore
7902	FOZIA	ANALYST	7566	30000	0	20	Karachi
7934	MURAD	CLERK	7782	13000	0	10	Islamabad

Task 1:
A software company is designing its internal employee management system.
The system stores **department information separately** from employee records to avoid data redundancy. Each employee is assigned to a department using a **department number**, but **department names are not stored directly in the employee table**. Before employee records can be stored successfully, the system must ensure that **only valid departments are referenced**.
If a department does not already exist, the employee record should not be allowed.

Your Tasks

1. Design a table to store department information with **exactly two attributes**:
   - One attribute to uniquely identify each department.
   - One attribute to store the name of the department (Name of Dep like Accountant, Sales and IT).
2. Insert records for the departments currently operating in the organization so that:
   - Each department has a **unique identifier**.
   - These identifiers can later be used when inserting employee records.
3. Create the employee table such that:
   - Each employee belongs to one of the departments defined earlier.

- The department identifier used for employees must already exist in the department table.

4. Insert employee records only after ensuring that the department data has been stored correctly.

Task 2:
The company database has Employee and Department tables. A new employee should only be added if their department already exists in the Department table. You need to write a query that inserts a new employee **only when the department is present**, otherwise the insertion should be ignored. This ensures that no employee is added with an invalid department.

Task 3:
The company database contains Employee and Department tables. You need to insert a new employee, but the insertion should happen **only if the department exists** in the Department table. For example, trying to add an employee with DepNo = 99 should be ignored if department 99 does not exist. Students must figure out how to write a query that conditionally inserts the employee.

After Creating both Task 2 and 3 queries student will note down what is the difference in their output result and what is logic behind that.

Task 4:
The company database has an Employee table that stores each employee's name and their sales figures. You need to retrieve the names and sales of employees whose sales are **less than 10,000**. Students must write a query that filters records based on a condition.

Task 5:
The company wants to list all employees working specifically in the 'Sales' department. Students must determine how to find employees based on department information, even when multiple departments exist.

Task 6:
Management wants to identify employees who are currently not assigned to any valid department. Students need to figure out how to detect employees without proper departmental allocation.

After Creating both Task 5 and 6 queries student will note down what is the difference in their output result and what is logic behind that.

Task 7:
The company wants to reward high-performing employees whose sales are above the team average. Students must find a way to compare individual performance with the overall performance.

Task 8:
Management wants to recognize the top-selling employee(s) in the company. Students must determine how to identify employees who have the highest sales, considering there may be ties.

Task 9:
The HR department wants to find employees who are linked to departments that no longer exist. Students must figure out how to detect employees with invalid department references.

Task 10:
The company wants to ensure all employees are assigned to valid departments. Students need to find a way to verify which employees have correct departmental assignments.

After Creating both Task 9 and 10 queries student will note down what is the difference in their output result and what is logic behind that.

Task 11:
Management wants to identify sales employees who are not earning any commission. Students must determine how to find employees based on both their role and their earnings, ensuring that only those who meet both conditions are selected.

Task 12:
The company wants to list employees who are **not part of the Research department**. Students need to figure out how to filter employees based on department assignments, even when multiple departments exist and ensure only the relevant employees are included.

Task 13:
The organization wants to clean up employee records that are assigned to departments that no longer exist. Students need to determine which records are invalid and how to remove them without affecting valid data.

NOTE FOR STUDENTS:

- Each group must write their own SQL queries. Copying from others or online is not allowed.
- Print all tasks and attach them to your lab manual.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric

Lab # 8 **Lab Title :** Implementation of MySQL Constraints and Relationships **Section :** I

Areas Assessed			Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)			Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)			Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)			Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr #	Maximum Marks ►		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Implementation of Different Types of Joins

OBJECTIVES

1. To understand the concept of SQL Joins
2. To implement different types of joins using SQL queries
3. To learn how data from multiple tables can be combined using joins

THEORETICAL BACKGROUND

The SQL **JOIN** clause is used to combine records from **two or more tables** in a database.
A join works on a **common column** present in both tables.

The joining of tables is actually performed using the **WHERE clause** or **JOIN condition**, which matches rows based on related column values.

WHY JOINS ARE IMPORTANT?

- SQL joins are important because they allow data to be retrieved from multiple tables at the same time.
- They help connect tables using common columns and show related data together.
- Joins make it easier to understand relationships between different tables.
- They help maintain data accuracy by avoiding unnecessary data duplication.
- Joins are essential for working with relational databases.

WHAT IS THE PURPOSE OF JOINS?

- The purpose of joins is to combine related data from two or more tables.
- Joins are used to retrieve meaningful and complete information from a database.
- They help generate reports by combining data stored in different tables.
- Joins support complex queries involving conditions across multiple tables.
- They allow efficient data analysis in database systems like MySQL.

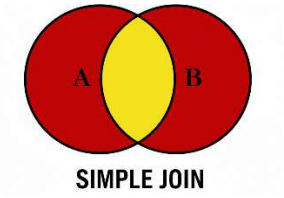
TYPES OF SQL JOINS

1. Simple Join
2. Self Join
3. Inner Join
4. Natural Join
5. Left Outer Join

6. Right Outer Join
7. Full Outer Join
8. Cross Join

1. SIMPLE JOIN

Simple join is the most common type of join.
It retrieves records from two tables that have **matching values** in a common column.

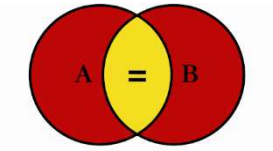


Types of Simple Join

- Equi Join
- Non-Equi Join

1. Equi Join

An equi join uses the **equal (=) operator** to match records between tables.



Example

```
SELECT *
FROM item, cust
WHERE item.id = cust.id;
```

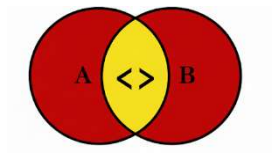
- This query returns only those rows where **item.id** and **cust.id** are equal.
- It combines matched rows from both tables.

Uses of Equi Join

- Insert records into another table
- Create tables with selected data
- Update records
- Create views

2. Non-Equi Join

A non-equi join uses **relational operators other than (=)** such as <, >, <=, >=.



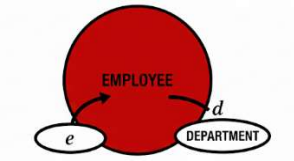
Example

```
SELECT *
FROM item, cust
WHERE item.id < cust.id;
```

This query matches records using the **less than (<)** operator.

3. Table Aliases

Table aliases are used to make SQL queries **shorter and easier to read**.

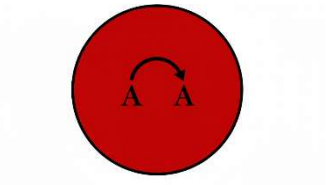


Example

```
SELECT *
FROM employee e, department d
WHERE e.depno = d.depno;
```

2. SELF JOIN

- A self join is a join where a table is joined **with itself**.
- It is used to compare rows within the same table.



Example

```
SELECT *
FROM emp x, emp y
WHERE x.salary > y.salary
AND x.deptno = y.deptno;
```

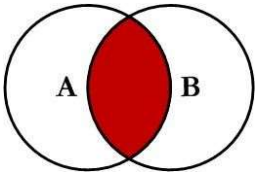
The same table is used twice with different aliases.

3. INNER JOIN

An **INNER JOIN** returns only the records that have **matching values in both tables**.

Syntax

```
SELECT *
FROM TableA INNER JOIN TableB
ON TableA.PK = TableB.PK;
```

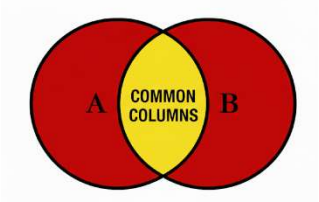


- Most commonly used join
- Returns only matching rows

Inner join can also use comparison operators like <, >, <>.

4. NATURAL JOIN

A **NATURAL JOIN** is an inner equi-join where SQL automatically matches columns with the **same name** in both tables.



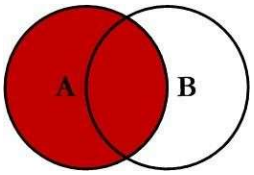
Example

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

- No need to specify join condition
- Automatically joins on common columns

5. LEFT OUTER JOIN

A **LEFT OUTER JOIN** returns:



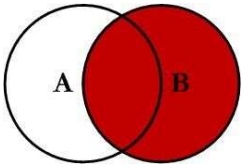
- All records from **left table**
- Matching records from **right table**
- NULL values where no match exists

Syntax

```
SELECT *
FROM TableA LEFT OUTER JOIN TableB
ON TableA.PK = TableB.PK;
```

6. RIGHT OUTER JOIN

A **RIGHT OUTER JOIN** returns:



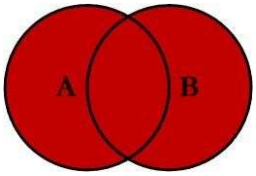
- All records from **right table**
- Matching records from **left table**
- NULL values where no match exists

Syntax

```
SELECT *
FROM TableA RIGHT OUTER JOIN TableB
ON TableA.PK = TableB.PK;
```

7. FULL OUTER JOIN

A **FULL OUTER JOIN** returns:



- All records from **both tables**
- Matching rows where available
- NULL where no match exists

Syntax

```
SELECT *
FROM TableA FULL OUTER JOIN TableB
ON TableA.PK = TableB.PK;
```

Full Outer Join in MySQL

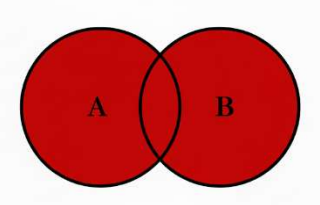
MySQL does **not support FULL OUTER JOIN directly**. It can be simulated using **UNION**:

```
SELECT *
FROM TableA LEFT JOIN TableB
ON TableA.PK = TableB.PK
UNION
SELECT *
FROM TableA RIGHT JOIN TableB
ON TableA.PK = TableB.PK;
```

8. CROSS JOIN

A **CROSS JOIN** produces a **Cartesian product**, meaning:

- Every row of Table A is combined with every row of Table B.



Syntax

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

OR

```
SELECT *
FROM TableA, TableB;
```

- No WHERE condition used
- Results in large output

SAMPLE EXAMPLES

Sample Tables

TableA

PK	Value
1	FOX
2	COP
3	TAXI
4	LINCOLN
5	ARIZONA
6	WASHINGTON
7	DELL

TableB

PK	Value
1	TROT
2	CAR
3	CAB
6	MONUMENT
7	PC
8	MICROSOFT
9	APPLE

1. Inner Join Example

Query

```
SELECT TableA.PK, TableA.Value, TableB.Value
FROM TableA INNER JOIN TableB
ON TableA.PK = TableB.PK;
```

Output

PK	TableA.Value	TableB.Value
1	FOX	TROT
2	COP	CAR
3	TAXI	CAB
6	WASHINGTON	MONUMENT
7	DELL	PC

Explanation:
An **INNER JOIN** returns only those records that have matching values in both tables.
If a row in TableA does not have a corresponding match in TableB, it will not appear in the result.
This join is mostly used when we want **common data from both tables only**.

2. Natural Join Example

Query

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

Output

PK	Value (TableA)	Value (TableB)
1	FOX	TROT
2	COP	CAR
3	TAXI	CAB
6	WASHINGTON	MONUMENT
7	DELL	PC

Explanation:
A **NATURAL JOIN** automatically joins two tables using columns that have the **same name and data type** in both tables.
There is no need to write the join condition explicitly.

3. Left Outer Join Example

Query

```
SELECT TableA.PK, TableA.Value, TableB.Value
FROM TableA LEFT OUTER JOIN TableB
ON TableA.PK = TableB.PK;
```

Output

PK	TableA.Value	TableB.Value
1	FOX	TROT
2	COP	CAR
3	TAXI	CAB
4	LINCOLN	NULL
5	ARIZONA	NULL
6	WASHINGTON	MONUMENT
7	DELL	PC

Explanation:
A **LEFT OUTER JOIN** returns **all records from the left table** and only the matching records from the right table.
This join is used when we want to keep all data from the left table.

4. Right Outer Join Example

Query

```
SELECT TableA.PK, TableA.Value, TableB.Value
FROM TableA RIGHT OUTER JOIN TableB
ON TableA.PK = TableB.PK;
```

Output

PK	TableA.Value	TableB.Value
1	FOX	TROT
2	COP	CAR
3	TAXI	CAB
6	WASHINGTON	MONUMENT
7	DELL	PC
8	NULL	MICROSOFT
9	NULL	APPLE

Explanation:
A **RIGHT OUTER JOIN** returns **all records from the right table** and only the matching records from the left table.
This join is useful when the right table is more important.

5. Full Outer Join Example

Query

```
SELECT *
FROM TableA LEFT JOIN TableB
ON TableA.PK = TableB.PK
UNION
SELECT *
FROM TableA RIGHT JOIN TableB
ON TableA.PK = TableB.PK;
```

Output

PK	TableA.Value	TableB.Value
1	FOX	TROT
2	COP	CAR
3	TAXI	CAB
4	LINCOLN	NULL
5	ARIZONA	NULL
6	WASHINGTON	MONUMENT
7	DELL	PC
8	NULL	MICROSOFT
9	NULL	APPLE

Explanation:
A **FULL OUTER JOIN** returns **all records from both tables**.
Since MySQL does not support full outer join directly, it is implemented using a **combination of LEFT JOIN and RIGHT JOIN with UNION**.

6. Cross Join Example

Query

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

Output (Sample)

TableA.Value	TableB.Value
FOX	TROT
FOX	CAR
FOX	CAB
COP	TROT
COP	CAR
...	...

Explanation:
A **CROSS JOIN** produces a **Cartesian product**, which means every row of the first table is combined with every row of the second table.
It does not require any join condition.
This join can generate a very large result set and should be used carefully.

LAB TASK

Create a table EMPLOYEE And DEPARTMENT with following schema:

TABLE A:

EmpNo	EmpName	Designation	OfficeNo	Sales	Commission	DepNo	City
7369	SALMAN	CLERK	7902	8000	0	20	Lahore
7499	ALIA	SALESMAN	7698	16000	3000	30	Karachi
7521	WAHEED	SALESMAN	7698	12500	5000	30	Lahore
7566	JAMIL	MANAGER	7839	29750	0	20	Islamabad
7654	MEHVISH	SALESMAN	7698	12000	12500	30	Karachi
7698	BILAL	MANAGER	7839	28500	0	30	Lahore
7782	QASIM	MANAGER	7839	24500	0	10	Islamabad
7788	SHAMS	ANALYST	7566	30000	0	20	Lahore
7839	KAMRAN	PRESIDENT	7788	50000	0	10	Karachi
7844	TANVEER	SALESMAN	7698	15000	0	30	Lahore
7876	AFTAB	CLERK	7788	11000	0	20	Islamabad
7900	JAVERIA	CLERK	7698	9500	0	30	Lahore
7902	FOZIA	ANALYST	7566	30000	0	20	Karachi
7934	MURAD	CLERK	7782	13000	0	10	Islamabad

TABLE B:

DepNo	DepName	Location
10	Accounts	Islamabad
20	Research	Lahore
30	Sales	Karachi
40	IT	Lahore

Task 1:
An organization maintains employee records separately from departmental information. Management wants a consolidated view that shows employees along with the departments they belong to, but only where a valid relationship exists between the two records. Employees whose department details are missing should not appear in the report. The purpose of this report is to ensure data consistency between employee assignments and department records. Students are required to retrieve information by correctly relating both datasets.

- *Prepare a query that ensures **only matched records from both datasets** are shown.*

Task 2:
The human resource department needs a focused report for performance evaluation. Only employees working in the Sales unit should be included, while all others must be excluded from the results. The report should clearly display each employee's name along with their job role. This information will be used to assess staffing within a specific department, requiring students to apply filtering after establishing relationships between tables.

- *Design a query that extracts **department-specific employees using relational logic**.*

Task 3:
As part of an office restructuring plan, management needs to know where employees are physically located. Each employee must be linked with the location of the department they work in. Employees without a valid departmental association should not be included in the output. The report aims to help management understand workforce distribution across different locations by combining employee and department information.

- *Write a query that combines employee records with **department location details**.*

Task 4:
The company wants a complete list of all employees for audit purposes. Some employees may not yet be assigned to any department, but they should still appear in the report. Where department information exists, it should be displayed alongside the employee name. This scenario helps identify unassigned or newly hired employees while still preserving available department details.

- *Create a query that prioritizes **all employees first**, then links department data.*

Task 5:
Management is reviewing deployment of employees across departmental locations. They require a report that lists every employee, regardless of whether a department has been assigned or not. If a department exists, its location should be shown; otherwise, employee details should still be visible. The aim is to ensure no employee record is omitted during analysis.

- *Write a query that ensures **no employee record is lost**.*

Task 6:
The organization follows a hierarchical structure where employees report to other employees acting as managers. Management wants a report that clearly shows each employee along with the name of their direct manager. Since both employees and managers are stored in the same dataset, students must logically relate records within the same table to produce the required output.

- *Design a query that relates employees **to themselves logically**.*

Task 7:
Sales performance is being evaluated across the organization. Management wants to identify employees whose sales figures exceed those of their respective managers. Only employees with a valid reporting relationship should be included in the analysis. The report should clearly present the employee's name, their sales amount, and the manager's identity for comparison.

- *Write a query that compares performance **within the reporting hierarchy**.*

Task 8:
The HR department is reviewing reporting discipline within departments. They want to ensure that employees and their managers belong to the same department. Only those reporting relationships where both individuals are part of the same department should appear in the results. This report helps in identifying any irregular cross-department reporting structures.

- *Prepare a query that verifies **both hierarchy and department consistency**.*

Task 9:
Management requires a comparative analysis of employee sales performance. The goal is to list pairs of employees where one employee has achieved higher sales than another. The comparison should not rely on equality but strictly on performance difference. This analysis will help in ranking employees based on sales achievements.

- *Write a query that forms **comparative employee pairs based on performance**.*

Task 10:
To identify exceptional performers, leadership wants a report showing employees whose sales are greater than their manager's sales. Only direct reporting relationships are valid for this comparison. The report should include both employee and manager sales figures to support decision-making for rewards and promotions.

- *Create a query that performs **hierarchical comparison using conditions**.*

Task 11:
The finance department is interested in high-performing sales staff. They want to identify employees who have achieved sales above a specific threshold and also view their respective managers for approval purposes. Employees who do not report to a manager should not be included. This report supports performance-based financial planning.

- *Write a query that filters **high sales performers with reporting details**.*

Task 12:
The training department is conducting a hypothetical analysis for planning future programs. They want to see every possible combination of employees and departments, regardless of actual assignments. This information will be used purely for simulation and planning purposes, without applying any matching conditions.

- *Design a query that produces **all possible combinations**.*

Task 13:
Facilities management is planning future office arrangements. They want to associate each employee with every department location to analyze possible relocation scenarios. Actual employee-department relationships are not relevant for this task. The objective is to generate all potential employee and location combinations.

- *Write a query that generates **complete employee–location combinations**.*

Task 14:
Administration is preparing a regional workforce report. They require details of employees who are based in Lahore along with the departments they belong to. Only employees with valid department assignments should be included. This report will assist management in understanding city-wise departmental distribution.

- *Prepare a query that combines **city-based filtering with relational data**.*

Task 15:
The HR department is compiling a city-specific employee summary. They want information about employees working in Karachi, including their names, job designations, and department titles. Only properly assigned employees should appear in the report. This data will support strategic planning and resource allocation decisions.

- *Write a query that extracts **multi-attribute information for a specific city**.*

NOTE FOR STUDENTS:

- Each group must write their own SQL queries. Copying from others or from online source is not allowed.
- Print all tasks and attach them to your lab manual.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric

Lab # 9 **Lab Title :** Implementation of different types of Joins **Section :** I

Areas Assessed			Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)			Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)			Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)			Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr#	Maximum Marks ▶		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Implementation of Group by Having Clause & Order by Clause

OBJECTIVES

1. To understand group (aggregate) functions in SQL.
2. To learn the use of GROUP BY clause.
3. To understand the use of HAVING with GROUP BY.
4. To apply JOIN with GROUP BY and ORDER BY clauses.
5. To sort and summarize data using ORDER BY.

THEORETICAL BACKGROUND

In SQL, **clauses** are different components of a query that define **what data to retrieve** and **how to process it**. Each clause performs a specific function in a SQL statement and helps in controlling the output of the query. Clauses are written in a fixed logical order and work together to retrieve meaningful and organized information from database tables.

Different Types of clauses are mentioned here:

- **SELECT clause** is used to choose the columns to display.
- **FROM clause** specifies the table from which data is retrieved.
- **WHERE clause** filters records before grouping.
- **GROUP BY clause** groups rows with similar values.
- **HAVING clause** filters grouped data using aggregate functions.
- **ORDER BY clause** sorts the final result in ascending or descending order.

WHAT IS THE PURPOSE OF CLAUSE?

- **To select required data** from one or more database tables.
- **To filter records** based on specific conditions.
- **To group similar records** for applying aggregate functions.
- **To filter grouped results** using the HAVING clause.
- **To sort data** in a meaningful order using ORDER BY.
- **To combine data from multiple tables** using JOIN clauses.
- **To generate summarized and organized reports** from large datasets.

1. GROUP FUNCTION

Group functions are used to perform calculations on a set of values and return a single result. These functions are mostly used with the **GROUP BY** clause.

Common group functions include:

- **SUM()** – Calculates total value
- **COUNT()** – Counts number of rows
- **AVG()** – Calculates average
- **MIN()** – Finds minimum value
- **MAX()** – Finds maximum value

These functions help in analyzing large amounts of data efficiently.

2. GROUP BY CLAUSE

The **GROUP BY** clause is used to group rows that have the same values in specified columns. It is commonly used with aggregate functions to generate summary results for each group.

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name;
```

Example

```
SELECT EmpNo, SUM(Salary)
FROM Emp
GROUP BY EmpNo;
```

Explanation

- The table is grouped based on **EmpNo**
- **SUM(Salary)** calculates total salary for each employee

Expected Output

EmpNo	SUM(Salary)
101	45000
102	52000
103	30000

Each employee number appears once with its total salary.

3. GROUP BY HAVING CLAUSE

The **HAVING** clause is used to apply conditions on grouped data.
It is similar to **WHERE**, but **WHERE cannot be used with aggregate functions**, therefore HAVING is required.

Order of Execution

- WHERE → GROUP BY → HAVING → ORDER BY

Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

Example

```
SELECT LastName, COUNT(OrderID) AS NumberOfOrders
FROM Orders
GROUP BY LastName
HAVING COUNT(OrderID) > 10;
```

Explanation

- Records are grouped by **LastName**
- **COUNT(OrderID)** counts total orders
- HAVING filters only those employees who have more than 10 orders

Expected Output

LastName	NumberOfOrders
Khan	15
Ali	12

Only employees meeting the HAVING condition are displayed.

4. JOIN USING GROUP BY CLAUSE

This query combines records from two tables using a common field and then groups the data to apply aggregate functions.

Syntax

```
SELECT fields_from_both_tables
FROM table1, table2
```

```
WHERE table1.common_field = table2.common_field
GROUP BY field_name;
```

Example

```
SELECT EmpNo, SUM(Salary)
FROM Emp, Dept
WHERE Emp.DeptNo = 20
GROUP BY EmpNo;
```

Explanation

- Data is joined from **Emp** and **Dept**
- Only employees of department 20 are selected
- Salary is summed for each employee

Expected Output

EmpNo	SUM(Salary)
201	40000
202	47000

5. ORDER BY CLAUSE

The **ORDER BY** clause is used to sort the result set in ascending or descending order based on one or more columns.

Default order is **Ascending (ASC)**.

Syntax

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

Example

```
SELECT EmpNo, EName, Job
FROM Emp
ORDER BY Job;
```

Explanation

Records are sorted alphabetically based on **Job**

Expected Output

EmpNo	EName	Job
103	Sana	Analyst
101	Ali	Clerk
102	Ahmed	Manager

6. JOIN USING ORDER BY CLAUSE

This query joins two tables and sorts the output based on a selected field.

Syntax

```
SELECT fields_from_both_tables
FROM table1, table2
WHERE table1.common_field = table2.common_field
ORDER BY field_name;
```

Example

```
SELECT EmpNo, EName, Job, DName
FROM Emp, Dept
WHERE Emp.DeptNo = 20
ORDER BY Job;
```

Explanation

- Employee and department data is joined
- Output is ordered based on job title

Expected Output

EmpNo	EName	Job	DName
301	Hina	Analyst	Sales
302	Bilal	Clerk	Sales

LAB TASK

Create a table EMPLOYEE And DEPARTMENT with following schema:

TABLE A:

EmpNo	EmpName	Designation	OfficeNo	Sales	Commission	DepNo	City
7369	SALMAN	CLERK	7902	8000	0	20	Lahore
7499	ALIA	SALESMAN	7698	16000	3000	30	Karachi
7521	WAHEED	SALESMAN	7698	12500	5000	30	Lahore
7566	JAMIL	MANAGER	7839	29750	0	20	Islamabad
7654	MEHVISH	SALESMAN	7698	12000	12500	30	Karachi
7698	BILAL	MANAGER	7839	28500	0	30	Lahore
7782	QASIM	MANAGER	7839	24500	0	10	Islamabad
7788	SHAMS	ANALYST	7566	30000	0	20	Lahore
7839	KAMRAN	PRESIDENT	7788	50000	0	10	Karachi
7844	TANVEER	SALESMAN	7698	15000	0	30	Lahore
7876	AFTAB	CLERK	7788	11000	0	20	Islamabad
7900	JAVERIA	CLERK	7698	9500	0	30	Lahore
7902	FOZIA	ANALYST	7566	30000	0	20	Karachi
7934	MURAD	CLERK	7782	13000	0	10	Islamabad

TABLE B:

DepNo	DepName	Location
10	Accounts	Islamabad
20	Research	Lahore
30	Sales	Karachi
40	IT	Lahore

Task 1:

Management wants to analyze sales performance at department level instead of individual employees. They want sales of all employees working in the same department to be **combined department-wise** and shown as one value. After this summary, departments should be arranged so that the highest performing department appears first. Since data needs to be **summarized for each department**, students should think about **GROUPING records before sorting**.

Task 2:

The administration department is preparing an employee list showing employee name, department name, and city. Employee data and department data are stored separately, so records must be **combined first**. The final output should be arranged city-wise for easy reading. Although this task focuses more on combining and sorting data, students should still understand how grouping differs from simple listing.

Task 3:

HR wants to know how many employees are working in each department. Individual employee details are not required; instead, employees should be **counted department-wise**. Each department should appear only once with its total number of employees. Since employees are being **counted for every department**, this task clearly requires **GROUPING data by department**.

Task 4:

Top management is only interested in departments that have a large workforce. They want to see departments where the **total number of employees per department** is greater than a given limit. Departments with fewer employees should be excluded. Students must first **group employees department-wise**, calculate totals, and then apply a condition on the grouped result.

Task 5:

The finance department wants to identify departments contributing high revenue. They need department names along with **total sales calculated for each department**. Only departments whose combined sales cross a target value should appear. Since sales must be **added for each department and then filtered**, students should use **GROUPING before applying conditions**.

Task 6:

Sales managers want to compare average performance of departments. They are not interested in totals but want to know the **average sales of employees in each department**. Each department should appear once with its calculated average. This clearly indicates the need to **group employees department-wise before calculating the average**.

Task 7:

Management wants a comparison report showing **average sales of each department** in ascending order. Department names must be shown, so data from multiple tables is needed. Since averages are required **for each department**, students must **group records by department** and then arrange the result.

Task 8:

The accounts department is reviewing commission expenses across cities. They want to know the **total commission paid in each city**, not individual payments. Each city should appear once with its combined commission amount. This task clearly involves **grouping employees city-wise before calculating totals**.

Task 9:

Management wants to focus only on high-performing cities. They need a list of cities where **total sales of all employees in that city** exceed a certain value. Cities with lower sales should not appear. Students should first **group sales by city**, calculate totals, and then apply a condition on those grouped values.

Task 10:

The sales manager wants to review employees working in the Sales department only. Employees should be listed with their sales values, and the list should be sorted from highest to lowest sales. This task mainly focuses on filtering and ordering, but students should also be able to distinguish it from **group-based summaries**.

Task 11:

Management wants to identify departments showing strong average performance. Only departments whose **average sales per department** are above a certain limit should be displayed, and higher averages should appear first. Since averages are calculated **per department**, students must apply **GROUPING before filtering and sorting**.

Task 12:

The company wants to see the **highest sale achieved in each department**. They do not want totals or averages, only one maximum value per department. Department names must be shown clearly. This requires employees to be **grouped department-wise before finding the maximum value**.

Task 13:

Regional management is reviewing sales generated by employees working in Islamabad. They want to see departments that have employees from this city and calculate **total sales department-wise for Islamabad only**. Departments whose totals exceed a given value should be shown. Students need to **filter first, then group by department and finally apply a condition**.

Task 14:

HR wants to analyze workforce strength based on job roles. They want to know how many employees fall under each designation. Each designation should appear once with its employee count, arranged from highest to lowest. Since employees are being **counted role-wise**, students must **group records by designation**.

NOTE FOR STUDENTS:

- Each group must write their own SQL queries. Copying from others or from online source is not allowed.
- Print all tasks and attach them to your lab manual.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric

Lab # 10 **Lab Title :** Implementation of Group By having clause & Order by clause **Section :** I

Areas Assessed			Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)			Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)			Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)			Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr#	Maximum Marks ►		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Structured Data Retrieval and Implementation

Using SQL JOINs and ORDER BY Clause

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric

Lab # 11 (OEL) **Lab Title :** Structured Data Retrieval and Implementation Using SQL JOINs and ORDER BY Clause **Section :** I

Areas Assessed			Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total				Total
			(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)					
Poor (0 to 30%)			Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4	
Satisfactory (31 to 70%)			Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork					
Very Good (71 to 100%)			Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work					
Sr.#	Maximum Marks ►		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													

Implementation of CASE Statements in SQL for Conditional Logic

OBJECTIVES

By the end of this lab, students should be able to:

1. Understand the concept of **conditional logic** in SQL.
2. Learn to implement **CASE statements** in queries.
3. Generate **dynamic output** based on conditions.
4. Apply CASE statements to **real-world scenarios** like grading, pass/fail status and distinctions.

THEORETICAL BACKGROUND

In SQL, sometimes we need to **change output or values based on certain conditions**. It works like an **IF-ELSE statement in programming**, allowing us to categorize, grade or classify data dynamically.

Why CASE is useful:

- Assign grades based on marks
- Classify employees based on salary
- Highlight special conditions (bonus, status)
- Conditional updates in tables

WHAT IS THE PURPOSE OF CASE STATEMENTS?

- Assign categories based on conditions (e.g., grades A, B, C, Fail).
- Update table columns dynamically depending on other column values (e.g., student status).
- Show different results for different conditions in a single query.
- Sort or rank data based on specified conditions.
- Replace multiple queries or joins with a single query using conditional logic.
- Used in grading, bonus calculations, status reports, performance analysis and classification tasks.

TYPES OF CASE STATEMENTS IN SQL

There are **two main types** of CASE statements:

1. Simple CASE Statement

Compares a column or expression to **specific values**.

Syntax:

```
CASE column_name
   WHEN value1 THEN result1
   WHEN value2 THEN result2
   ELSE result_default
END
```

Example:

```
SELECT StudentName, Marks,
CASE Marks
   WHEN 85 THEN 'Excellent'
   WHEN 72 THEN 'Good'
   ELSE 'Average'
END AS Performance
FROM Student;
```

2. Searched CASE Statement

- Evaluates **conditions** rather than exact values.
- More flexible than simple CASE.

Syntax:

```
CASE
   WHEN condition1 THEN result1
   WHEN condition2 THEN result2
   ELSE result_default
END
```

Example:

```
SELECT StudentName, Marks,
CASE
   WHEN Marks >= 80 THEN 'A'
   WHEN Marks >= 60 THEN 'B'
   ELSE 'C'
END AS Grade
FROM Student;
```

The **searched CASE** is most commonly used for grading, pass/fail status or multiple conditions.

METHODS OF USING CASE STATEMENTS IN SQL

1. Using CASE in SELECT Statements

The most common use of CASE is in a **SELECT query** to modify the output dynamically.

Explanation:

- Evaluate each row against conditions
- Return a **custom value** based on the result

Example: Grading Students

```
SELECT StudentName, Marks,
CASE
   WHEN Marks >= 80 THEN 'A'
   WHEN Marks >= 60 THEN 'B'
   WHEN Marks >= 50 THEN 'C'
   ELSE 'Fail'
END AS Grade
FROM Student;
```

Output:

StudentName	Marks	Grade
Ali	85	A
Sara	72	B
Ahmed	45	Fail
Ayesha	90	A
Zain	55	C

2. Using CASE for Status Classification

CASE can categorize data into **status groups** such as Pass/Fail/Distinction.

Example:

```
SELECT StudentName, Marks,
CASE
   WHEN Marks >= 80 THEN 'Distinction'
   WHEN Marks >= 50 THEN 'Pass'
   ELSE 'Fail'
END AS Status
FROM Student;
```

Output:

StudentName	Marks	Status
Ali	85	Distinction
Sara	72	Pass
Ahmed	45	Fail
Ayesha	90	Distinction
Zain	55	Pass

Explanation:

Students are dynamically classified based on **marks thresholds**.

3. Using CASE in UPDATE Statements

CASE can be used to **update table data conditionally**.

Explanation:

Useful to set values dynamically based on criteria without multiple UPDATE commands.

Example: Update Student Status

```
ALTER TABLE Student ADD COLUMN Status VARCHAR(20);

UPDATE Student
SET Status = CASE
   WHEN Marks >= 80 THEN 'Distinction'
   WHEN Marks >= 50 THEN 'Pass'
   ELSE 'Fail'
END;
```

Output (after update):

StudentID	StudentName	Marks	Status
1	Ali	85	Distinction
2	Sara	72	Pass
3	Ahmed	45	Fail
4	Ayesha	90	Distinction
5	Zain	55	Pass

Explanation:

Dynamically updates **Status** column for all students in one query.

4. Using CASE in ORDER BY

CASE can control **sorting order dynamically**.

Example: Sort Students by Status

```
SELECT StudentName, Marks
FROM Student
ORDER BY CASE
   WHEN Marks >= 80 THEN 1
   WHEN Marks >= 50 THEN 2
   ELSE 3
END;
```

Effect:

Students with **Distinction** appear first, then **Pass** then **Fail**.

5. Using CASE for Conditional Calculations

CASE can also be used for **calculating values dynamically**, such as bonuses, discounts, or rewards.

Example: Assign Bonus Based on Marks

```
ALTER TABLE Student ADD COLUMN Bonus INT;

UPDATE Student
SET Bonus = CASE
   WHEN Marks >= 80 THEN 1000
   WHEN Marks >= 60 THEN 500
   ELSE 0
END;
```

Explanation:

Students with higher marks receive higher bonus automatically.

LAB TASK

TABLE:

EmpNo	EmpName	Designation	OfficeNo	Sales	Commission	DepNo	City
7369	SALMAN	CLERK	7902	8000	0	20	Lahore
7499	ALIA	SALESMAN	7698	16000	3000	30	Karachi
7521	WAHEED	SALESMAN	7698	12500	5000	30	Lahore
7566	JAMIL	MANAGER	7839	29750	0	20	Islamabad
7654	MEHVISH	SALESMAN	7698	12000	12500	30	Karachi
7698	BILAL	MANAGER	7839	28500	0	30	Lahore
7782	QASIM	MANAGER	7839	24500	0	10	Islamabad
7788	SHAMS	ANALYST	7566	30000	0	20	Lahore
7839	KAMRAN	PRESIDENT	7788	50000	0	10	Karachi
7844	TANVEER	SALESMAN	7698	15000	0	30	Lahore
7876	AFTAB	CLERK	7788	11000	0	20	Islamabad
7900	JAVERIA	CLERK	7698	9500	0	30	Lahore
7902	FOZIA	ANALYST	7566	30000	0	20	Karachi
7934	MURAD	CLERK	7782	13000	0	10	Islamabad

Task 1

The management wants to evaluate employee performance based on their sales. Each employee should be assigned a **performance category** such as Excellent, Good, Average, or Below Average depending on their sales amount. This will help identify top performers quickly and give feedback to those who need improvement. Students should think about **applying conditional logic to assign categories dynamically** based on sales.

Task 2

The HR department wants to **calculate bonuses for employees based on the commission they have earned**. Employees with higher commissions should get higher bonus amounts, while those with little or no commission get smaller or zero bonus. This task requires students to **use conditional logic to assign values based on ranges** ensuring that the correct bonus is applied to each employee.

Task 3

The company wants to classify employees according to their **job levels or hierarchy**. **For example**, **Presidents are part of Top Management, Managers are Middle Management, Analysts are Specialists, and so on**. Each employee's designation must be checked and labeled appropriately. Students should understand how to **map specific values to categories using exact matches in conditional logic.**

Task 4

Management wants to identify top performers in different cities. Since sales targets vary by location, an employee may be considered a top performer in one city but not in another. **For each employee, a city-specific performance status should be assigned**. Students should think about **combining multiple conditions within a single evaluation** to handle city-wise thresholds.

Task 5

HR has added a new column called **Status** in the employee table. Now, every employee's status should be set dynamically based on their sales figures. Top performers get labels like Star Employee or High Performer, while others are flagged as Needs Improvement. Students should use **conditional logic in an UPDATE statement** to change data based on existing values.

Task 6

Management wants to generate a **sorted list of employees based on performance levels**. Employees should appear in order from highest to lowest performance, with tie-breakers handled by their sales figures. This requires assigning a **numeric rank or sort order** dynamically and then using it to sort the table. Students should understand **how to apply conditional logic for sorting purposes**.

Task 7

The company wants to determine **eligibility for a special allowance**. Salesmen are eligible if their commission is above a certain amount, and Managers are eligible if their sales exceed a threshold. All others are not eligible. Students should learn to **combine multiple conditions in a single evaluation** to produce a status column showing eligibility.

NOTE FOR STUDENTS:

- Each group must write their own SQL queries. Copying from others or from online source is not allowed.
- Print all tasks and attach them to your lab manual.

Rubric Based Assessment of CLO

IET217L Database Systems Lab Evaluation Rubric													
Lab # 12		Lab Title : Implementation of CASE Statements in SQL for Conditional Logic					Section : I						
Areas Assessed		Database Functionality	Database Organization	Ability to Debug or Refine Design	Explanation of Concepts and Results	Organization of Experimental Results	Demonstrates professional, ethical, responsible behavior.	CLO wise Total					Total
		(CLO 1)	(CLO 1)	(CLO 1)	(CLO 2)	(CLO 3)	(CLO 4)						
Poor (0 to 30%)		Database not functional or contains major errors	Poorly structured tables missing relationships	Unable to identify or fix issues	Unable to explain database design or queries	Results not organized or unclear	Unable to demonstrate professionalism or teamwork	CLO 1	CLO 2	CLO 3	CLO 4		
Satisfactory (31 to 70%)		Database partially functional queries execute with some errors	Partial schema correctness incomplete relationships	Debugged and refined with help	Partially explains design logic and query purpose	Partially organized missing labels or explanations	Demonstrates partial responsibility and teamwork						
Very Good (71 to 100%)		Database fully functional queries execute correctly without errors	Well-structured schema with correct relationships and normalization	Independently debugged and refined successfully	Clearly explains database design and SQL implementation logic	Results well-organized with proper output and explanation	Demonstrates full professionalism, ethical behavior and teamwork during lab work						
Sr#	Maximum Marks ►		2	1	2	2	2	1	5	2	2	1	10
	Student ID ▼	Student Name ▼											
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													
15													
16													
17													
18													
19													
20													