## ****Day 1: SQL - Introduction & Table Management****

**Training Duration:** 9:45 AM – 6:00 PM

### ****Objective:****

Introduce the basics of SQL and RDBMS, focusing on table creation, data types, and constraints.

**9:45 AM – 11:00 AM: Introduction to SQL & RDBMS**

DBMS (Database Management System) is a broad category of software for managing databases, while RDBMS (Relational Database Management System) is a specific type of DBMS that organizes data into tables with predefined relationships. RDBMS is a subset of DBMS and is more structured and efficient for managing complex, interrelated data.

DBMS:

* **Data Storage:** Can store data in various formats, including files, hierarchical structures, or network models.
* **Relationships:** May or may not enforce relationships between data elements.
* **Examples:** File systems, XML databases, and some older database systems.
* **Use Cases:** Often used for smaller applications with simpler data structures and limited user access.

RDBMS:

* **Data Storage:** Organizes data into tables with rows (records) and columns (attributes).
* **Relationships:** Enforces relationships between tables using keys (primary and foreign keys), ensuring data integrity.
* **Examples:** MySQL, PostgreSQL, Oracle, SQL Server.
* **Use Cases:** Suitable for applications requiring complex data relationships, high data integrity, and multiple users.
* **Key Features:** Supports ACID properties (Atomicity, Consistency, Isolation, Durability) for reliable transaction processing.

Key Differences Summarized:

|  |  |  |
| --- | --- | --- |
| **Feature** | **DBMS** | **RDBMS** |
| Data Storage | Files, hierarchical, network | Tables with rows and columns |
| Relationships | May not enforce | Enforces relationships using keys |
| Data Integrity | Less emphasis | High emphasis, uses constraints |
| Scalability | Limited scalability | Designed for scalability |
| User Support | Typically single user | Supports multiple users |
| Examples | File systems, XML databases | MySQL, PostgreSQL, Oracle, SQL Server |
| ACID Properties | Not always enforced | Enforces ACID properties |

#### ****What is SQL?****

Structured Query Language (SQL) is the standard language used to manage and manipulate relational databases. SQL allows you to insert, update, delete, and query data from a database. It also provides capabilities for schema creation, data access control, and transaction management.

| **Aspect** | **SQL (Relational)** | **NoSQL (Non-relational)** |
| --- | --- | --- |
| **Data Structure** | Tables with rows and columns | Document-based, key-value, column-family, or graph-based |
| **Schema** | Fixed schema (predefined structure) | Flexible schema (dynamic and adaptable) |
| **Scalability** | Vertically scalable (upgrading hardware) | Horizontally scalable (adding more servers) |
| **Data Integrity** | ACID-compliant (strong consistency)  Robust (Strong) | BASE- Basically available compliant (more available, less consistent) – Simple and |
| **Query Language** | SQL (Structured Query Language) | Varies (e.g., MongoDB uses its own query language) |
| **Performance** | Efficient for complex queries and transactions | Better for large-scale data and fast read/write operations |
| **Use Case** | Best for transactional systems (banking, ERP, etc.) | Ideal for big data, real-time web apps, and data lakes |
| **Examples** | MySQL, PostgreSQL, Oracle, MS SQL Server | MongoDB, Cassandra, CouchDB, Neo4j |

#### ****What is RDBMS?****

RDBMS stands for Relational Database Management System. It is a type of database system that stores data in a structured format using rows and columns. Popular RDBMS systems include Oracle, MySQL, SQL Server, and PostgreSQL.

**Key Features of RDBMS:**

* Data is stored in tables (relations)
* Each table has a primary key
* Tables can be linked using foreign keys
* Ensures data integrity and reduces redundancy

**Why We Use SQL & RDBMS in Organizations:**

* To manage large-scale structured data
* Centralized access to real-time data
* Ensures accuracy and consistency
* Supports integration with enterprise applications
* Provides role-based access and control

|  |  |
| --- | --- |
| SQL | PL/SQL |
| Essentially, SQL is a structured language used to query and manipulate data stored in databases. | It is a procedural language that is designed to implement SQL statements in a better way. |
| The query executes a single operation at a time. | Group of operations is performed in a single block. |
| SQL is declarative (program specifies what is to be done than how it is to be done). | PL/SQL is procedural (code that is written as a sequence of instructions). |
| SQL is used in relational databases to execute various queries like create table, delete table, insert into table, etc. | PL/SQL is used to write program blocks, procedures, functions, cursors, triggers, and packages. |
| SQL does not support data variables. | PL/SQL provides support for variable constraints and data types. |
| Control structures are not supported. | Control structures such as if-else, For loop, While loop are supported. |
| SQL is used to retrieve data from the database. We can modify data and table structure with SQL. | PL/SQL is used to create web applications and server pages. |
| It is possible to embed SQL in PLSQL syntax as PLSQL is an extension of SQL. | Embedding PL/SQL in SQL syntax is not possible. |
| SQL directly interacts with the database server. | PL/SQL does not directly interact with the database server. |
| Error handling feature is not present in SQL. | PL/SQL handles errors and exceptions effectively with the help of the inbuilt exception handlers. |
| Handling a large chunk of data can’t be effectively achieved by SQL. | PL/SQL handles a large chunk of data effectively with the help of procedures, functions and triggers. |
| When dealing with voluminous data, SQL offers a slower processing speed. | PL/SQL offers a high processing speed |
| Execution of statements in SQL does not result in reduced traffic as multiple statements cannot get executed at the same time. The queries are executed one at a time which increases the network traffic. | Execution of operation results in reduced network traffic as a block of statements is executed at once. |
| SQL is easy to use and understand. | Certain concepts of PL/SQL can be complex and prior knowledge might be necessary. |
| SQL does not support I/O operations. | PL/SQL supports I/O operations as it can accept inputs and then store and process it. |
| In SQL, we use both DML (Data Manipulation Language) and DDL (Data Definition Language) to write queries and commands. | The PL/SQL language, on the other hand, is composed of code blocks containing triggers, functions, variables, conditional statements (if..then..else), and control structures (for loops, whiles). |
| The SQL language allows for the execution of a single operation or query at a time. | PL/SQL, however, allows the execution of entire blocks or multiple operations at once. This reduces network traffic. |

Yes — **PL/SQL is considered an advanced extension of SQL**, but with a clear distinction: **SQL is a query language**, while **PL/SQL is a full procedural programming language** designed to work with SQL.

"**SQL** is a declarative language — it tells the database what to do (e.g., SELECT, INSERT, UPDATE), but it cannot control how it's done with logic like loops or conditions.

**PL/SQL**, on the other hand, is procedural — it allows us to write complex logic, control flow, declare variables, use loops, IF-ELSE, create procedures/functions/packages, and even handle exceptions.

Think of **PL/SQL as the programming layer built on top of SQL** — enabling automation, modularity, and advanced business rules right within the database."

Comparison Table

| **Feature** | **SQL** | **PL/SQL** |
| --- | --- | --- |
| Type | Declarative | Procedural |
| Used For | Querying & manipulating data | Automating tasks, writing logic, managing control flow |
| Control Structures | No (Only Queries) | IF, WHILE, FOR, CASE |
| Variables | No | Yes |
| Error Handling | Not possible | EXCEPTION block |
| Object-Oriented Logic |  | Procedures, Functions, Packages |
| Use Case | SELECT employees from a department | Build a billing procedure with discount logic & exception handling |

### ****Real-Life Analogy (Optional for Experienced Learners)****

**SQL is like giving one-time instructions to a robot.**  
**PL/SQL is like programming the robot with logic to make decisions, repeat tasks, and respond to situations (errors, inputs, etc.).**

While both **Oracle PL/SQL** and **MySQL** support stored procedures, functions, and triggers, **PL/SQL is a full-featured procedural programming language**, whereas **MySQL’s procedural capabilities are more limited** and are **not equivalent in depth or optimization to PL/SQL**.

### ****Detailed Explanation (Use This in Class)****

| **Feature** | **Oracle PL/SQL** | **MySQL Stored Programs** |
| --- | --- | --- |
| **Language Maturity** | Very rich, matured procedural language since Oracle 7 | Basic support added in MySQL 5.0 (2005), less mature |
| **Packages** | Supports **Packages** (modular unit of Procedures, Functions, Variables) | No concept of packages |
| **Exception Handling** | Very robust, with **named and user-defined exceptions** | Basic exception handling (DECLARE HANDLER) |
| **Cursors** | Advanced support for **explicit, implicit, parameterized cursors** | Available but limited |
| **Triggers** | BEFORE, AFTER, ROW-LEVEL, STATEMENT-LEVEL | Supported but with fewer options |
| **Collections (Arrays)** | Supports VARRAY, Nested Tables, Associative Arrays | Not supported |
| **Advanced Features** | Dynamic SQL (DBMS\_SQL), Role-based logic, Bulk Collect, FORALL | Mostly not available |
| **Compiler** | Has a PL/SQL compiler; supports optimization and debugging | No full compiler for procedural code |

"Yes, MySQL has some procedural programming features like stored procedures and triggers. However, **PL/SQL is a much richer and enterprise-grade procedural language**, designed for large-scale systems with modularity, reusable packages, robust error handling, and advanced performance optimization like **bulk operations**.

PL/SQL is widely used in **banking, healthcare, telecom, ERP systems**, etc., where high performance and strict logic control are essential. If you're coming from a MySQL background, you'll find PL/SQL has much more to offer .

#### ****Types of SQL Commands:****

* **DDL (Data Definition Language):** CREATE, ALTER, DROP
* **DML (Data Manipulation Language):** INSERT, UPDATE, DELETE
* **DCL (Data Control Language):** GRANT, REVOKE
* **TCL (Transaction Control Language):** COMMIT, ROLLBACK, SAVEPOINT

"Think of SQL as the language we use to talk to databases. It helps us manage the structure of data, manipulate the data itself, control who has access, and ensure transactional consistency. This is foundational in all enterprise software systems like payroll, HRMS, CRM, etc."

**Code Examples:**

-- Creating a table

CREATE TABLE Employee (

EmpID INT PRIMARY KEY,

Name VARCHAR(100),

DepartmentID INT,

Email VARCHAR(100) UNIQUE,

Salary NUMBER(8,2) CHECK (Salary > 0),

JoiningDate DATE NOT NULL

);

-- Department Table

CREATE TABLE Department (

DeptID INT PRIMARY KEY,

DeptName VARCHAR(50) UNIQUE

);

-- Adding a Foreign Key

ALTER TABLE Employee

ADD CONSTRAINT fk\_dept FOREIGN KEY (DepartmentID) REFERENCES Department(DeptID);

**11:00 AM – 11:15 AM: Tea/Coffee Break**

**11:15 AM – 12:45 PM: CREATE, ALTER, DROP, and Data Types**

**SQL Data Types:**

* **VARCHAR(n):** Variable length character data
* **NUMBER(p,s):** Numeric data with precision and scale
* **DATE:** Date and time values
* **CHAR(n):** Fixed length character

**PL/SQL, namely scalar types**, **numeric types, character**types, **Boolean** types, date and time types, large objects, and a type by user-defined subtypes.

**Scalar Data Types in PL/SQL**

Scalar data types are basic types that store only one value at a time.like numbers, characters, or even logical values each representing an individual value. The scalar data types are categorized into:

* **Numeric Types:** It Stores any integer value
* **Character Types**: represent strings of text
* **Boolean Types**: It Contains ‘**true**’ or ‘**false**’ values.
* **Datetime Types:** It isUsed to represent date and time values which are of typical usage in computer systems.

Subtypes are defined based on the base scalar types and are formed by placing additional constraints upon values that can be assigned.

**Numeric Data Types and Subtypes in PL/SQL**

Numeric data types store numbers, both integers and real numbers, and allow developers to perform arithmetic operations. The main numeric types include:

* **NUMBER**: A highly flexible type that can store fixed-point or floating-point numbers. It has precision and scale parameters. For example, **NUMBER(5, 2)**can store up to 5 digits, with 2 of them after the decimal point.
* **FLOAT**: It is a subtype of NUMBER designed for storing floating-point numbers. You can specify an optional precision such as **FLOAT(10)** which allows for storing a number with up to 10 digits of precision

**Character Data Types**

Character data types are designed to store any text, numbers or symbols in the form of alphanumeric. They are used specifically for string manipulation and are a vital part of PL/SQL.

* **CHAR**: It Handles variable-length binary data and fixed-length character strings. If the string is less than the defined length, then the rest is filled up with spaces. For instance, CHAR(10) would store string as CHAR data type that has a length of 10 characters regardless of the actual string length.
* **VARCHAR2**: To store character strings of varying lengths. **VARCHAR2** is slightly different because it only allocates the required amount of space required to store the string. For instance,**VARCHAR2(10)** data type can accommodate a string with as many as 10 characters.
* **LONG**: It Can store variable-length character strings of up to 2 gigabytes. However, it is deprecated and it should be replaced with **CLOB**to 2 GB. However, it is deprecated and should be avoided in favour of CLOB.

**Subtypes of Character Data Types**

* **STRING**: A subtype of VARCHAR2, used to represent variable-length strings.
* **LONG VARCHAR**: A deprecated subtype of VARCHAR2, previously used to store large strings.

**PL/SQL Boolean Data Types**

The Boolean data type is unique to PL/SQL, allowing you to store logical values and use them in conditional expressions.

**BOOLEAN**: This type can have three possible values: TRUE, FALSE, or NULL. It is used in conditional statements and logical comparisons. Notably, the BOOLEAN data type is unique to PL/SQL and cannot be used in SQL statements directly.

**PL/SQL Datetime and Interval Types**

Datetime data types contain date and time and interval types contain the difference between two datetime values. PL/SQL provides the following datetime and interval types:

* **DATE:**It stores **date** and **time** values. These are the year, month, day, hour, minute, and second as a part of the Date type.
* **TIMESTAMP**: It is an extension of the DATE data type with the added feature of fractional seconds.
* **TIMESTAMP WITH TIME ZONE**: Saves a TIMESTAMP, but with information about the time zone in which it has been set.
* **TIMESTAMP WITH LOCAL TIME ZONE**: Standalone function that converts the TIMESTAMP to the time zone of the current database session.
* **INTERVAL YEAR TO MONTH:** Saves the amount of time measured in years and months.
* **INTERVAL DAY TO SECOND**: Saves as the time duration in terms of days, hours, minutes, and seconds.

**PL/SQL Large Object (LOB) Data Types**

LOB data types store large amounts of unstructured data, such as text, images, videos, and audio. PL/SQL provides several LOB types:

* **BLOB(Binary Large Object)**: Stores binary large objects, such as images or multimedia files.
* **CLOB(Character Large Object)**: Stores large character data.
* **NCLOB(National Character Large Object)**: Stores large character data character set.
* **BFILE(Binary File)**: Stores a reference to a binary file stored outside of the database.

LOB types can store up to 4 gigabytes of data, making them ideal for handling large and complex datasets.

**PL/SQL Data Types: Examples for Each Category**

**1. Scalar Data Types**

**Numeric Types**

DECLARE

v\_salary NUMBER(8,2); -- Can store values like 123456.78

v\_rating FLOAT(4); -- Can store floating-point numbers

BEGIN

v\_salary := 85000.75;

v\_rating := 4.5;

END;

*Used for salaries, scores, quantities, etc.*

**Character Types**

DECLARE

v\_emp\_name CHAR(10); -- Fixed-length string

v\_emp\_code VARCHAR2(10); -- Variable-length string

BEGIN

v\_emp\_name := 'Alice';

v\_emp\_code := 'E102';

END;

*Used for storing names, codes, departments.*

**Boolean Type**

DECLARE

v\_is\_active BOOLEAN;

BEGIN

v\_is\_active := TRUE;

IF v\_is\_active THEN

DBMS\_OUTPUT.PUT\_LINE('Employee is active.');

END IF;

END;

*Used for conditional logic in PL/SQL blocks.*

DECLARE

v\_employee\_id NUMBER := 102;

v\_first\_name VARCHAR2(50) := 'Jane';

v\_last\_name VARCHAR2(50) := 'Smith';

v\_salary NUMBER := 60000;

BEGIN

INSERT INTO employees (employee\_id, first\_name, last\_name, salary)

VALUES (v\_employee\_id, v\_first\_name, v\_last\_name, v\_salary);

COMMIT;

END;

**Date and Time Types**

DECLARE

v\_join\_date DATE;

v\_updated\_at TIMESTAMP;

v\_offset\_time TIMESTAMP WITH TIME ZONE;

BEGIN

v\_join\_date := SYSDATE;

v\_updated\_at := SYSTIMESTAMP;

v\_offset\_time := CURRENT\_TIMESTAMP;

END;

*Used for storing join dates, logs, and timestamps.*

**What Are Interval Types in PL/SQL?**

**Interval types** are used to store **durations** — not specific timestamps or dates, but *periods of time*, like:

* “2 years and 3 months”
* “5 days, 12 hours, 30 minutes, and 15 seconds”

These are useful when you want to **add or subtract time** from a date or just **store time durations**.

**Types of Interval Types in Oracle PL/SQL**

There are two main types of intervals:

| **Interval Type** | **Description** |
| --- | --- |
| INTERVAL YEAR TO MONTH | Represents a period in **years and months** |
| INTERVAL DAY TO SECOND | Represents a period in **days, hours, minutes, seconds** |

**Interval Types**

DECLARE

v\_duration1 INTERVAL YEAR TO MONTH := INTERVAL '2-3' YEAR TO MONTH;

v\_duration2 INTERVAL DAY TO SECOND := INTERVAL '5 12:30:15' DAY TO SECOND;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Duration 1: ' || v\_duration1);

DBMS\_OUTPUT.PUT\_LINE('Duration 2: ' || v\_duration2);

END;

*Used to represent durations or time differences.*

DECLARE

v\_duration1 INTERVAL YEAR TO MONTH := INTERVAL '2-3' YEAR TO MONTH;

v\_duration2 INTERVAL DAY TO SECOND := INTERVAL '5 12:30:15' DAY TO SECOND;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Duration 1: ' || v\_duration1);

DBMS\_OUTPUT.PUT\_LINE('Duration 2: ' || v\_duration2);

END;

**Explanation:**

* v\_duration1 is storing an interval of:
  + **2 years and 3 months**
* v\_duration2 is storing an interval of:
  + **5 days, 12 hours, 30 minutes, 15 seconds**

**Expected Output:**

Duration 1: +02-03

Duration 2: +05 12:30:15.000000

Note: The format includes:

* +02-03 → +YY-MM for INTERVAL YEAR TO MONTH
* +05 12:30:15.000000 → +DD HH:MI:SS.FFFFFF for INTERVAL DAY TO SECOND

**Where You Can Use This**

You can use these interval values:

* To **add/subtract durations** from a DATE
* To calculate **project timelines**
* To store **time gaps** between activities (e.g., patient discharge time – admission time)

**Example:**

DECLARE

v\_start\_date DATE := DATE '2024-01-01';

v\_new\_date DATE;

BEGIN

v\_new\_date := v\_start\_date + INTERVAL '1-2' YEAR TO MONTH; -- Add 1 year, 2 months

DBMS\_OUTPUT.PUT\_LINE('New Date: ' || v\_new\_date);

END;

**Output:**

New Date: 01-MAR-2025

**2. Large Objects (LOB) Types**

DECLARE

v\_resume CLOB;

v\_profile\_pic BLOB;

v\_contract BFILE;

BEGIN

-- Assume proper DB directory and LOB initialization here

NULL;

END;

*Used for storing documents, images, external files.*

**3. User-Defined Subtypes**

DECLARE

SUBTYPE salary\_type IS NUMBER(8,2);

v\_new\_salary salary\_type;

BEGIN

v\_new\_salary := 92000.50;

DBMS\_OUTPUT.PUT\_LINE('Salary: ' || v\_new\_salary);

END;

*Subtypes restrict base types to improve readability, consistency, and safety.*

**Use Case Example – HR Employee Table Script**

CREATE TABLE Employee (

EmpID NUMBER(5) PRIMARY KEY,

EmpName VARCHAR2(50) NOT NULL,

Gender CHAR(1) CHECK (Gender IN ('M', 'F')),

Email VARCHAR2(100) UNIQUE,

JoinDate DATE,

Salary NUMBER(8,2) CHECK (Salary > 0),

Resume CLOB,

ProfilePic BLOB

);

*This table uses:*

* NUMBER, VARCHAR2, DATE, CHAR
* CLOB & BLOB for large text and image
* UNIQUE, NOT NULL, CHECK, PRIMARY KEY constraints

**Table Structure Commands:**

* **CREATE:** Define new table structure
* **ALTER:** Modify an existing table (add, modify, drop columns)
* **DROP:** Delete table permanently

**Code Examples:**

-- Altering the Employee table

ALTER TABLE Employee ADD PhoneNumber VARCHAR(15);

ALTER TABLE Employee MODIFY Salary NUMBER(10,2);

**12:45 PM – 1:30 PM: Lunch Break**

**1:30 PM – 3:30 PM: Constraints in SQL**

**Constraints Overview:**

* **PRIMARY KEY:** Uniquely identifies each record in a table. Only one per table.
* **FOREIGN KEY:** Enforces referential integrity by linking to the primary key of another table.
* **UNIQUE:** Ensures all values in a column are different.
* **NOT NULL:** Ensures that a column cannot store NULL values.
* **CHECK:** Restricts the values that can be placed in a column based on a condition.

**Single Table Example with All Constraints:**

CREATE TABLE Patient (

PatientID INT PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Email VARCHAR(100) UNIQUE,

Age INT CHECK (Age >= 0),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Department(DeptID)

);

-- Adding a Foreign Key

ALTER TABLE Employee

ADD CONSTRAINT fk\_dept FOREIGN KEY (DepartmentID) REFERENCES Department(DeptID);

-- Altering the Employee table

ALTER TABLE Employee ADD PhoneNumber VARCHAR(15);

ALTER TABLE Employee MODIFY Salary NUMBER(10,2);

**Use Case: Hospital Employee Management System**

In a hospital setup:

* Every **Employee** must have a unique ID → PRIMARY KEY
* Employees must belong to a valid **Department** → FOREIGN KEY
* Each **Email** must be unique → UNIQUE
* **Salary** must be greater than zero → CHECK
* **Joining Date** must always be captured → NOT NULL

This ensures data reliability, referential integrity, and correctness. These constraints reduce data anomalies such as duplicate records or invalid references.

**3:30 PM – 3:45 PM: Tea/Coffee Break**

**3:45 PM – 5:45 PM: Lab Practice on Table Creation & Constraints**

**Lab Task:** Create the following tables using at least 3 constraints on each:

1. **Employee**
2. **Department**
3. **Patient**
4. **Project**

CREATE TABLE Project (

ProjectID INT PRIMARY KEY,

ProjectName VARCHAR(100) NOT NULL,

StartDate DATE,

EndDate DATE CHECK (EndDate > StartDate)

);