UNIT 1 – DATABASE CONCEPTS

1. Introduction to Database Systems

A database is an organized collection of related data that is stored electronically. It enables efficient retrieval, insertion, and modification of data. A Database Management System (DBMS) is software designed to store, manage, and provide access to databases in a secure and structured manner.
Examples include Oracle, MySQL, SQL Server, PostgreSQL, and MongoDB.

A database allows multiple users to share data while ensuring consistency, integrity, and security. It eliminates redundancy and provides a centralized control of data, which helps in effective decision-making and business operations.

2. Information and Data

- Data refers to raw facts or figures that may not have any meaning on their own (e.g., numbers, names, or dates).
- Information is processed or structured data that carries meaning and supports decision-making.

For example:

- *Data:* 101, Sharmila, MCA
- *Information:* “Student with Roll Number 101 named Sharmila is pursuing MCA.”

In databases, data is stored in tables, and information is derived by querying and processing that data.

3. File System and Its Problems

Before the advent of databases, organizations used traditional file-based systems to store information. Each application maintained its own files, often in text or binary format.

Problems with File Systems

1. Data Redundancy: Same data stored in multiple files.
2. Data Inconsistency: Different copies of the same data may not match.
3. Lack of Data Isolation: Data scattered in various files makes retrieval difficult.
4. Difficulty in Access: No standard way to query data.
5. Integrity Problems: No constraints to ensure data accuracy.
6. Security Issues: Limited control over user access.
7. Concurrency Issues: Multiple users cannot easily access or update the same file simultaneously.

To overcome these limitations, database systems were introduced.

4. Database Systems

A Database Management System (DBMS) is a collection of interrelated data and a set of programs to access that data.
It provides:

- A data definition language (DDL) for defining database structures.
- A data manipulation language (DML) for querying and modifying data.
- Transaction management, concurrency control, and backup/recovery features.

Advantages of DBMS:

1. Data Integration: Centralized data management.
2. Reduced Redundancy: Shared data storage.
3. Improved Data Integrity: Enforces consistency.
4. Security and Privacy: Controlled access using permissions.
5. Data Independence: Changes in data structure do not affect applications.
6. Concurrent Access and Recovery: Supports multiple users and protects data from failure.

---

5. Data Models

A data model is a conceptual framework that describes the structure of data, relationships, and constraints. It acts as a blueprint for designing databases.

Types of Data Models

1. Hierarchical Model:
   Data is organized in a tree-like structure. Each child has only one parent.
   Example: IBM's Information Management System (IMS).
2. Network Model:
   Data is represented using records and links. Each record can have multiple parents and children.
3. Relational Model:
   Data is stored in tables (relations) consisting of rows (tuples) and columns (attributes).
   Example: SQL-based systems like Oracle, MySQL.
4. Object-Oriented Model:
   Combines database concepts with object-oriented programming. Objects store both data and behavior (methods).
5. Entity-Relationship (ER) Model:
   Uses entities, attributes, and relationships to represent data in a diagrammatic form.
6. Document/NoSQL Models:
   Used in modern big-data systems like MongoDB, where data is stored as key-value pairs or documents.

6. Importance of Data Models

- Provides a conceptual framework for database design.
- Defines data relationships, integrity rules, and constraints.
- Ensures that data is stored and retrieved efficiently.

- Helps developers, users, and administrators understand the data structure clearly.

7. Basic Building Blocks of a Database

1. Entity: A real-world object or concept (e.g., Student, Employee).
2. Attribute: Properties that describe an entity (e.g., Name, Age).
3. Relationship: Association between entities (e.g., Student *enrolls in* Course).
4. Constraint: Rules that ensure data validity (e.g., Roll number must be unique).
5. Schema: Logical design of the database.
6. Instance: Actual data stored in the database at a particular moment.

8. Business Rules

Business rules define policies, procedures, or constraints in an organization that influence database design.
Example:

- Each student must have a unique registration number.
- An employee cannot approve his own leave.

These rules ensure the database enforces organizational logic and supports accurate transactions.

9. Evolution of Data Models

Data models have evolved over time to meet the growing complexity of data management:

Generation	Model Type	Description
1st	File-based System	Independent files with no structure
2nd	Hierarchical & Network Models	Structured storage with parent-child relationships
3rd	Relational Model	Based on tables and mathematical set theory
4th	Object-Oriented Model	Integration with programming objects
5th	NoSQL & Big Data Models	Supports unstructured and distributed data

This evolution reflects the movement from rigid, structure-dependent models to flexible, scalable, and distributed systems.

10. Degrees of Data Abstraction

DBMS provides multiple levels of abstraction to separate the way data is viewed by users from how it is actually stored.

1. Physical Level:
   Describes how data is physically stored on hardware. Example: files, indexing, storage blocks.

2. Logical Level:
   Describes what data is stored and relationships between them. Example: tables, columns, constraints.
3. View Level:
   The highest level of abstraction that shows only a part of the database relevant to a specific user or application. Example: a student may see only their marks, not others'.

This multi-level structure ensures data independence and simplifies management.

UNIT 2 – DATABASE DESIGN CONCEPTS

1. Introduction to Relational Database Model

The Relational Database Model, proposed by E.F. Codd in 1970, represents data in the form of tables (known as *relations*). Each table consists of rows (tuples) and columns (attributes). This model provides a logical view of data and forms the foundation of most modern DBMSs such as Oracle, MySQL, and SQL Server.

Example:
A simple *STUDENT* table:

Student_ID	Name	Department	Age
S001	Rahul	CSE	20
S002	Priya	IT	21
S003	Karthik	ECE	22

Here,

- Table name: STUDENT
- Columns: Student_ID, Name, Department, Age
- Rows: Individual student records

2. Logical View of Data

The logical view defines *what data is stored* and *how the data elements relate to each other*, rather than *how they are physically stored*.
This abstraction helps users interact with the data using queries without worrying about its physical location or structure.

Example:
When a user writes SELECT Name FROM STUDENT; they only see the logical structure of the STUDENT table, not how it is stored in memory.

3. Keys in Relational Database

Keys play a crucial role in identifying and connecting records in relational tables.

Types of Keys:

1. Primary Key:
   A unique identifier for each record in a table.
   Example: *Student_ID* in STUDENT table.
2. Candidate Key:
   A set of attributes that can uniquely identify tuples. One of these becomes the primary key.
3. Alternate Key:
   Remaining candidate keys that are not chosen as the primary key.

4. Foreign Key:
   An attribute in one table that refers to the primary key of another table, establishing a relationship between them.
5. Composite Key:
   A key formed by combining two or more attributes.
   Example: *(Student_ID, Course_ID)* in an ENROLLMENT table.
6. Super Key:
   Any combination of attributes that uniquely identifies a record (includes candidate keys).

4. Integrity Rules

Integrity constraints ensure accuracy and consistency of data within a database.

1. Entity Integrity:
   Ensures that every table has a primary key and that the key's value is unique and not null.
2. Referential Integrity:
   Ensures that foreign key values always match existing values in the referenced table.
3. Domain Integrity:
   Ensures that data entered into a column is of the correct data type and range.

Example:
If *Department_ID* in EMPLOYEE table is a foreign key referencing the DEPARTMENT table, referential integrity ensures every EMPLOYEE has a valid department.

5. Relational Set Operators

Relational algebra provides operations that can be performed on tables (relations).

Operator	Description
UNION	Combines tuples from two relations, removing duplicates
UNION ALL	Combines tuples including duplicates
INTERSECT	Returns common tuples from two relations
MINUS (EXCEPT)	Returns tuples from one relation not present in another
CARTESIAN PRODUCT	Combines every row of one table with every row of another
JOIN	Combines related rows based on a condition

6. Data Dictionary and System Catalog

- Data Dictionary:
  A repository that contains metadata — information about data such as tables, columns, data types, and constraints.
- System Catalog:
  A part of the DBMS that stores detailed information about the database structure. It is automatically maintained by the system.

Example of metadata:

Table Name	Column Name	Data Type	Constraint
STUDENT	Student_ID	INT	Primary Key
STUDENT	Name	VARCHAR	NOT NULL

7. Data Redundancy Revisited

Data redundancy refers to storing the same piece of data in multiple places.
In relational databases, redundancy is minimized through normalization and relationships.

Example:
Instead of storing department names repeatedly in each employee record, we store the *Department_ID* and link it to a DEPARTMENT table.

8. Indexes

An index is a database structure that improves the speed of data retrieval operations.
It works like an index in a book — allowing faster access without scanning the entire table.

Types of Indexes:

1. Primary Index – Based on primary key.
2. Secondary Index – Based on non-primary attributes.
3. Clustered Index – Determines physical order of data.
4. Non-clustered Index – Maintains a separate structure from data.

9. Codd's Rules

Dr. E.F. Codd proposed 12 rules to define a true relational database system.
Some important ones are:

1. Information Rule: All data should be stored in tables.
2. Guaranteed Access: Each value can be accessed by table name, column name, and primary key.
3. Systematic Null Handling: The system must support nulls for missing or unknown information.
4. Data Independence: Changes in physical storage shouldn't affect the logical structure.
5. Integrity Independence: Integrity constraints must be stored in the catalog, not in application programs.

10. Entity-Relationship Model (ER Model)

The ER Model describes data using entities, attributes, and relationships. It helps in database design before implementation.

Components:

- Entity: Real-world object (e.g., Student, Course).

- Attribute: Property of entity (e.g., Name, Age).
- Relationship: Association between entities (e.g., Student enrolls in Course).

Types of Relationships:

1. One-to-One (1:1)
   Example: Each employee has one ID card.
2. One-to-Many (1:M)
   Example: One department has many employees.
3. Many-to-Many (M:N)
   Example: Students can enroll in multiple courses.

11. ER Diagram

An Entity Relationship Diagram (ERD) visually represents entities, attributes, and relationships.

Symbols used:

- Rectangle: Entity
- Oval: Attribute
- Diamond: Relationship
- Lines: Connect entities and attributes

Example:
STUDENT — ENROLLS — COURSE
(Each student can enroll in many courses, and each course can have many students)

UNIT 3 – NORMALIZATION OF DATABASE TABLES

1. Introduction

When a database is designed, raw data is often placed in large, unstructured tables. This leads to data redundancy, update anomalies, and inconsistencies.
To solve these issues, a process called Normalization is used.

Normalization is the process of organizing data into well-structured tables to minimize redundancy and improve data integrity.

2. Database Tables, Schema, and Normalization

- A table (relation) represents data in rows and columns.
- A schema defines the structure — table names, fields, and relationships.
- Normalization restructures these tables into smaller, related tables using a set of rules called Normal Forms.

Objectives of Normalization:

1. Eliminate redundant data.
2. Ensure data dependencies make sense.
3. Simplify database maintenance.
4. Avoid insertion, deletion, and update anomalies.

3. The Need for Normalization

Without normalization, a database suffers from:

- Data Redundancy – Same data repeated unnecessarily.
- Update Anomalies – Updating one record but forgetting others.
- Insertion Anomalies – Unable to add new data without other data.
- Deletion Anomalies – Losing valuable data when deleting records.

Example of Unnormalized Table:

Student_ID	Student_Name	Course	Instructor
S001	Rahul	DBMS	Meena
S001	Rahul	Java	Suresh

Here, the student's name "Rahul" is repeated, leading to redundancy.

4. The Normalization Process

Normalization is carried out in several stages, each called a Normal Form (NF).
Each form has rules that the database must satisfy before moving to the next.

1st Normal Form (1NF)

- Each column should contain atomic (indivisible) values.
- Each record should be unique.

Example (1NF Conversion):
Unnormalized: Course = {DBMS, Java}
→ Normalized: Separate each course into different rows.

Student_ID	Student_Name	Course
S001	Rahul	DBMS
S001	Rahul	Java

2nd Normal Form (2NF)

- Must first satisfy 1NF.
- All non-key attributes must depend entirely on the primary key (no partial dependency).

Example:

Student_ID	Course_ID Student_Name Course_Name
Here, *Student_Name* depends only on *Student_ID*, and *Course_Name* depends only on *Course_ID* — not on the combination.	

Solution:
Split the table into two:

1. STUDENT(Student_ID, Student_Name)
2. COURSE(Course_ID, Course_Name)

3rd Normal Form (3NF)

- Must first satisfy 2NF.
- Remove transitive dependency (non-key attributes depending on other non-key attributes).

Example:
| Student_ID | Student_Name | Dept_ID | Dept_Name |
Here, *Dept_Name* depends on *Dept_ID*, not directly on *Student_ID*.

Solution:
Create a new DEPARTMENT table:

1. STUDENT(Student_ID, Student_Name, Dept_ID)
2. DEPARTMENT(Dept_ID, Dept_Name)

Boyce-Codd Normal Form (BCNF)

- A stronger version of 3NF.

- For every functional dependency X → Y, X must be a super key.
  BCNF eliminates all redundancy caused by functional dependencies.

Higher Normal Forms (4NF and 5NF)

1. 4th Normal Form (4NF):
   Removes *multi-valued dependencies* (when one key is related to multiple independent attributes).
2. 5th Normal Form (5NF):
   Deals with *join dependencies* — ensures tables can be reconstructed from smaller ones without data loss.

These forms are rarely needed in practice but are important for theoretical understanding and highly complex databases.

5. Denormalization

Sometimes, for performance reasons, normalization is *partially reversed* — this is called Denormalization.
It combines multiple related tables to reduce the number of joins in queries, improving speed at the cost of redundancy.

Example:
Merging STUDENT and DEPARTMENT tables into one for faster access.

---

6. Advantages of Normalization

- Reduces redundancy and saves storage.
- Improves data consistency and integrity.
- Simplifies maintenance and updates.
- Enhances query performance for structured data.

---

7. Disadvantages of Over-normalization

- Too many small tables can slow down query execution due to excessive joins.
- More complex queries are needed to fetch combined data.
- May not be suitable for analytical systems or data warehouses.

---

8. Practical Example – Stepwise Normalization

Unnormalized Table:

Student_ID	Name	Courses	Instructor
S001	Priya	{DBMS, Java}	{Meena, Suresh}

1NF: Split repeating groups

Student_ID	Name	Course	Instructor
S001	Priya	DBMS	Meena
S001	Priya	Java	Suresh

2NF: Remove partial dependencies
→ Create separate tables

1. STUDENT(Student_ID, Name)
2. COURSE(Course, Instructor)

3NF: Remove transitive dependencies
→ Split further if Instructor depends on another attribute like Dept_ID.

UNIT 4 – ADVANCED SQL AND PL/SQL FUNDAMENTALS

1. Introduction to SQL

SQL (Structured Query Language) is a standardized language used to manage and manipulate relational databases.
It includes commands for defining, querying, and controlling data and access permissions.

SQL commands are categorized into:

1. DDL (Data Definition Language): Defines database structures.
   - Examples: CREATE, ALTER, DROP, TRUNCATE
2. DML (Data Manipulation Language): Manages data inside tables.
   - Examples: INSERT, UPDATE, DELETE, SELECT
3. DCL (Data Control Language): Controls access to data.
   - Examples: GRANT, REVOKE
4. TCL (Transaction Control Language): Manages database transactions.
   - Examples: COMMIT, ROLLBACK, SAVEPOINT

2. Data Definition Commands

Used to create and modify database structures.

Examples:

```
CREATE TABLE STUDENT (
   Student_ID INT PRIMARY KEY,
   Name VARCHAR(30),
   Department VARCHAR(20)
);
ALTER TABLE STUDENT ADD Age INT;
DROP TABLE STUDENT;
```

3. Data Manipulation Commands

Used to manage the data inside tables.

Examples:

```
INSERT INTO STUDENT VALUES (1, 'Ravi', 'CSE', 21);
UPDATE STUDENT SET Department='IT' WHERE Student_ID=1;
DELETE FROM STUDENT WHERE Student_ID=1;
```

4. SELECT Queries

The SELECT statement retrieves data from one or more tables.

Syntax:

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

Example:

SELECT Name, Department FROM STUDENT WHERE Age > 20;

Keywords in SELECT:

- DISTINCT – Removes duplicates.
- ORDER BY – Sorts results.
- GROUP BY – Groups rows with the same values.
- HAVING – Filters groups.
- WHERE – Filters rows before grouping.

5. Joining Database Tables

JOIN operations combine data from two or more tables based on related columns.

Types of Joins:

1. INNER JOIN:
   Returns rows with matching values in both tables.
2. SELECT s.Name, d.Dept_Name
3. FROM STUDENT s
4. INNER JOIN DEPARTMENT d ON s.Dept_ID = d.Dept_ID;
5. LEFT OUTER JOIN:
   Returns all rows from the left table, even if no match exists in the right table.
6. RIGHT OUTER JOIN:
   Returns all rows from the right table, even if no match exists in the left table.
7. FULL OUTER JOIN:
   Returns all rows when there is a match in either table.
8. CROSS JOIN:
   Returns the Cartesian product of both tables.
9. SELF JOIN:
   A table joined with itself.

6. SQL Relational Set Operators

Operator	Function
UNION	Combines results of two queries, removes duplicates
UNION ALL	Combines results and keeps duplicates
INTERSECT	Returns common rows between two queries
MINUS (EXCEPT)	Returns rows from the first query not found in the second

Example:

SELECT Name FROM STUDENT_CSE
UNION
SELECT Name FROM STUDENT_IT;

7. Subqueries and Correlated Queries

- Subquery: A query inside another query.
- Correlated Subquery: Depends on the outer query for its values.

Example:

```
SELECT Name FROM STUDENT
WHERE Dept_ID IN (SELECT Dept_ID FROM DEPARTMENT WHERE
Dept_Name='CSE');
```

8. SQL Clauses and Functions

Common Clauses:

- WHERE – Filters records.
- HAVING – Filters grouped records.
- GROUP BY – Groups similar records.
- ORDER BY – Sorts results ascending/descending.

Aggregate Functions:

- SUM(), AVG(), MAX(), MIN(), COUNT()

String Functions:

- UPPER(), LOWER(), LENGTH(), CONCAT(), SUBSTR()

Date Functions:

- SYSDATE, MONTHS_BETWEEN(), ADD_MONTHS()

Numeric Functions:

- ROUND(), CEIL(), FLOOR(), MOD()

9. Introduction to PL/SQL

PL/SQL (Procedural Language/SQL) is Oracle's procedural extension of SQL.
It allows embedding procedural logic like loops, conditions, and variables inside SQL blocks.

Advantages:

- Improves performance (block execution).
- Enhances security.
- Supports exception handling.
- Allows modular programming using procedures and functions.

10. Structure of PL/SQL Block

A PL/SQL block has three parts:

```
DECLARE
   -- Declarations
BEGIN
   -- Executable statements
EXCEPTION
   -- Error handling
END;
```

Example:

```
SET SERVEROUTPUT ON;
DECLARE
   v_name VARCHAR2(20) := 'Ravi';
BEGIN
   DBMS_OUTPUT.PUT_LINE('Hello ' || v_name);
END;
```

11. Data Types and Variables

Common PL/SQL data types:

- NUMBER, CHAR, VARCHAR2, DATE, BOOLEAN

Variable Declaration:

```
DECLARE
   student_name VARCHAR2(20);
   total_marks NUMBER(5);
```

12. Operators in PL/SQL

Type	Example
Arithmetic	+, -, *, /, **
Comparison	=, <>, >, <, >=, <=
Logical	AND, OR, NOT

13. Control Structures

PL/SQL supports conditional and iterative control statements.

IF Statement:

```
IF total_marks >= 50 THEN
   DBMS_OUTPUT.PUT_LINE('Pass');
ELSE
   DBMS_OUTPUT.PUT_LINE('Fail');
END IF;
```

LOOP Structures:

- LOOP...EXIT WHEN...END LOOP
- WHILE...LOOP...END LOOP
- FOR...LOOP...END LOOP

14. Nested Blocks

PL/SQL allows blocks inside blocks — these are nested blocks.
They help in variable scoping and modular code structure.

15. Embedded SQL

PL/SQL allows embedding SQL statements directly within the procedural code.

Example:

```
DECLARE
  v_name STUDENT.Name%TYPE;
BEGIN
  SELECT Name INTO v_name FROM STUDENT WHERE Student_ID = 101;
  DBMS_OUTPUT.PUT_LINE('Student Name: ' || v_name);
END;
```

16. Data Manipulation in PL/SQL

You can use SQL commands like INSERT, UPDATE, DELETE within PL/SQL blocks.

Example:

```
BEGIN
  INSERT INTO STUDENT VALUES (105, 'Kavya', 'ECE', 20);
  COMMIT;
END;
```

17. Transaction Control Statements

Transaction control ensures data consistency and error recovery.

Command	Description
COMMIT	Saves all changes permanently.
ROLLBACK	Undoes changes since the last commit.
SAVEPOINT	Marks a point to rollback partially.

Example:

```
SAVEPOINT before_update;
UPDATE STUDENT SET Dept='IT' WHERE Student_ID=105;
ROLLBACK TO before_update;
```

18. PL/SQL Cursors

Cursors are used to handle query results row by row.

Types:

1. Implicit Cursor: Automatically created for single-row queries.
2. Explicit Cursor: Declared manually for multi-row queries.

Example of Explicit Cursor:

```
DECLARE
  CURSOR c1 IS SELECT Name FROM STUDENT;
  v_name STUDENT.Name%TYPE;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO v_name;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_name);
  END LOOP;
  CLOSE c1;
END;
```

Unit 5 – PL/SQL (Procedural Language / Structured Query Language)

1. Introduction to PL/SQL

PL/SQL is an extension of SQL developed by Oracle to combine the power of SQL with the procedural capabilities of programming languages.
While SQL handles data manipulation, PL/SQL adds logic, loops, conditions, and modular programming.

Features of PL/SQL:

- Block-structured: Code is organized into blocks.
- Supports variables and constants.
- Exception handling for runtime errors.
- Portability: Runs on any Oracle database.
- Supports control structures (IF, LOOP, WHILE, etc.).
- Allows modular programming using procedures, functions, and packages.

2. PL/SQL Block Structure

Every PL/SQL program is divided into three main sections:

```
DECLARE
   -- Declaration section (variables, constants, cursors)
BEGIN
   -- Executable section (SQL statements, loops, conditions)
EXCEPTION
   -- Error handling section
END;
```

Example:

```
DECLARE
   v_name VARCHAR2(20);
BEGIN
   v_name := 'Ravi';
   DBMS_OUTPUT.PUT_LINE('Hello ' || v_name);
END;
```

Explanation:

- DECLARE – optional, used to define variables or constants.
- BEGIN – mandatory; holds executable statements.
- EXCEPTION – optional; handles errors.
- END; – marks the end of the block.

3. Variables and Data Types

Declaring Variables:

Syntax:

```
variable_name datatype [:= initial_value];
```

Example:

```
DECLARE
  emp_id NUMBER := 1001;
  emp_name VARCHAR2(30) := 'Arun';
BEGIN
  DBMS_OUTPUT.PUT_LINE(emp_id || ' - ' || emp_name);
END;
```

Common Data Types:

Data Type	Description
NUMBER	Numeric values
VARCHAR2	Variable-length character data
DATE	Date and time values
BOOLEAN	TRUE/FALSE
CHAR	Fixed-length character data

4. Operators in PL/SQL

Type	Operators
Arithmetic	+, -, *, /, **
Relational	=, <>, >, <, >=, <=
Logical	AND, OR, NOT
Concatenation	`

Example:

```
IF salary > 20000 THEN
  DBMS_OUTPUT.PUT_LINE('High Salary');
END IF;
```

5. Control Structures

Conditional Statements

IF...THEN

```
IF condition THEN
   statements;
END IF;
```

IF...THEN...ELSE

```
IF condition THEN
   statements1;
ELSE
   statements2;
END IF;
```

IF...ELSIF...ELSE

```
IF grade = 'A' THEN
   DBMS_OUTPUT.PUT_LINE('Excellent');
ELSIF grade = 'B' THEN
   DBMS_OUTPUT.PUT_LINE('Good');
ELSE
   DBMS_OUTPUT.PUT_LINE('Needs Improvement');
END IF;
```

Looping Constructs

LOOP

```
LOOP
   statements;
   EXIT WHEN condition;
END LOOP;
```

WHILE LOOP

```
WHILE counter <= 5 LOOP
   DBMS_OUTPUT.PUT_LINE(counter);
   counter := counter + 1;
END LOOP;
```

FOR LOOP

```
FOR i IN 1..5 LOOP
   DBMS_OUTPUT.PUT_LINE('Number: ' || i);
END LOOP;
```

6. Nested Blocks

PL/SQL allows blocks within blocks (inner and outer blocks).
Variables declared in the inner block override those in the outer block (scope concept).

Example:

```
DECLARE
  x NUMBER := 10;
BEGIN
  DECLARE
    x NUMBER := 5;
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Inner x: ' || x);
  END;
  DBMS_OUTPUT.PUT_LINE('Outer x: ' || x);
END;
```

7. SQL Statements in PL/SQL

You can use DML commands directly inside the PL/SQL block.

Example (INSERT):

```
BEGIN
  INSERT INTO employees VALUES(101, 'Kumar', 25000);
  DBMS_OUTPUT.PUT_LINE('Record Inserted');
END;
```

Example (UPDATE):

```
BEGIN
  UPDATE employees SET salary = 30000 WHERE emp_id = 101;
  DBMS_OUTPUT.PUT_LINE('Record Updated');
END;
```

Example (DELETE):

```
BEGIN
  DELETE FROM employees WHERE emp_id = 101;
  DBMS_OUTPUT.PUT_LINE('Record Deleted');
END;
```

---

8. Transaction Control Statements

Command	Description
COMMIT	Saves changes permanently
ROLLBACK	Undoes uncommitted changes
SAVEPOINT	Marks a point to roll back to

Example:

```
BEGIN
  INSERT INTO emp VALUES(1, 'Kavi');
  SAVEPOINT A;
  INSERT INTO emp VALUES(2, 'Ravi');
  ROLLBACK TO A;  -- Rolls back second insert only
  COMMIT;
END;
```

9. Exception Handling

Used to manage runtime errors and prevent program termination.

Syntax:

```
BEGIN
  -- statements
EXCEPTION
  WHEN exception_name THEN
    statements;
END;
```

Example:

```
BEGIN
  SELECT salary INTO v_sal FROM emp WHERE emp_id = 500;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Employee not found');
END;
```

10. Cursors

A cursor is a pointer that allows row-by-row processing of SQL results.

Types:

1. Implicit Cursor – automatically created for single SQL statements.

2. Explicit Cursor – defined manually by the user.

Example of Explicit Cursor:

```
DECLARE
  CURSOR emp_cur IS SELECT emp_name FROM employees;
  v_name employees.emp_name%TYPE;
BEGIN
  OPEN emp_cur;
  LOOP
    FETCH emp_cur INTO v_name;
    EXIT WHEN emp_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(v_name);
  END LOOP;
  CLOSE emp_cur;
END;
```

11. Advantages of PL/SQL

- Increases performance through block execution.
- Supports modular and reusable programming.
- Handles errors effectively.
- Allows interaction between SQL and procedural logic.
- Enhances security through stored procedures and triggers.