Module 4 – Introduction to DBMS

Introduction to SQL Theory

1. What is SQL, and why is it essential in database management?

SQL (Structured Query Language) is a standardized programming language used to manage and manipulate relational databases. It allows users to:

- Create and modify database structures (tables, views, indexes, etc.)
- Insert, update, delete, and retrieve data
- Control access and manage user permissions

Why it's essential:

- It provides a consistent way to interact with databases.
- Enables efficient querying and data manipulation.
- Is supported by all major relational database systems (like MySQL, PostgreSQL, SQL Server, and Oracle).

2. Explain the difference between DBMS and RDBMS.

Feature	DBMS (Database Management System)	RDBMS (Relational DBMS)
Data structure	Stores data as files or documents	Stores data in tables (rows and columns)
Relation support	No relational constraints between data	Enforces relationships via foreign keys
Normalization	Rarely supports normalization	Supports data normalization
Examples	Microsoft Access, XML DB	MySQL, PostgreSQL, Oracle, SQL Server
ACID compliance	Not guaranteed	Fully ACID-compliant (ensures data integrity)

3. Describe the role of SQL in managing relational databases.

SQL is the primary interface for working with relational databases. Its roles include:

- Defining the structure of database objects (DDL - Data Definition Language)
- Manipulating data within the tables (DML - Data Manipulation Language)
- Controlling user access and permissions (DCL - Data Control Language)
- Ensuring data consistency and integrity through constraints and transactions (TCL - Transaction Control Language)

Essentially, SQL is what allows users and applications to communicate with and control relational databases.

4. What are the key features of SQL?

- Data Querying: Retrieve data using SELECT statements.
- Data Manipulation: Add, update, and delete data using INSERT, UPDATE, and DELETE.
- Data Definition: Create and manage schemas, tables, and other structures (CREATE, ALTER, DROP).
- Data Control: Manage access with GRANT and REVOKE.
- Transaction Control: Ensure data integrity with COMMIT, ROLLBACK, and SAVEPOINT.
- Built-in Functions: Support for aggregation, string, and date/time functions.

2. SQL Syntax

Theory Questions:

1. What are the basic components of SQL syntax?

SQL syntax consists of a set of rules that define how SQL statements are written and interpreted. The basic components include:

- Keywords: Reserved words like SELECT, FROM, WHERE, INSERT, UPDATE, etc.
- Identifiers: Names of database objects like tables, columns, and indexes.
- Operators: Used in expressions, e.g., =, >, <, LIKE, IN, AND, OR.
- Literals: Constant values like strings ('John'), numbers (100), or dates ('2025-06-18').
- Expressions: Combinations of identifiers, operators, and literals that return a value.
- Clauses: Logical parts of a statement, like WHERE, GROUP BY, ORDER BY.
- Semicolon (;): Ends an SQL statement (especially in systems that allow multiple statements).

2. Write the general structure of an SQL SELECT statement.

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

Example:

```
SELECT name, salary
FROM employees
WHERE department = 'HR'
GROUP BY name
HAVING salary > 50000
ORDER BY salary DESC;
```

3. Explain the role of clauses in SQL statements.

Clauses are the building blocks of SQL statements. Each clause serves a specific purpose:

Clause	Role
SELECT	Specifies the columns to return.
FROM	Identifies the table(s) to retrieve data from.
WHERE	Filters rows based on conditions.
GROUP BY	Groups rows sharing a value into summary rows (used with aggregates).
HAVING	Filters groups created by GROUP BY.
ORDER BY	Sorts the result set by one or more columns.
JOIN	Combines rows from two or more tables based on a related column.

3. SQL Constraints

Theory Questions:

1. What are constraints in SQL?

Constraints in SQL are rules applied to table columns to enforce data integrity and ensure the accuracy and reliability of the data stored in the database.

They prevent invalid data entry and help maintain the logical consistency of the database.

Types of Constraints in SQL:

Constraint	Description
PRIMARY KEY	Uniquely identifies each row in a table. Cannot be NULL.
FOREIGN KEY	Ensures referential integrity by linking a column to the PRIMARY KEY in another table.
NOT NULL	Ensures that a column cannot have NULL values.
UNIQUE	Ensures all values in a column are distinct (no duplicates).
CHECK	Ensures that values in a column satisfy a specific condition.
DEFAULT	Sets a default value for a column when no value is provided.

2. How do PRIMARY KEY and FOREIGN KEY constraints differ?

Feature	PRIMARY KEY	FOREIGN KEY
Purpose	Uniquely identifies a row in a table	Establishes a relationship between two tables
Uniqueness	Must be unique and NOT NULL	Can contain duplicate values
Location	Defined in the owning table	Refers to a key in another table
Number per table	Only one PRIMARY KEY per table	Can have multiple FOREIGN KEYs in a table
Example	id column in employees table	dept_id in employees referencing departments.id

3. What is the role of NOT NULL and UNIQUE constraints?

NOT NULL:

- Ensures that a column must have a value.
- Prevents the insertion of NULLs into the column.
- Used when a field is mandatory, like usernames or email addresses.

UNIQUE:

- Ensures that all values in the column are distinct.
- Allows only one NULL (depending on the RDBMS).
- Useful for enforcing unique data, like national ID numbers or email addresses.

4. Main SQL Commands and Sub-commands (DDL)

Theory Questions:

1. Define the SQL Data Definition Language (DDL).

DDL (Data Definition Language) is a category of SQL commands used to define and manage database structures such as tables, schemas, indexes, and views.

Common DDL commands:

- CREATE – to create new database objects (tables, views, etc.)
- ALTER – to modify existing objects
- DROP – to delete objects
- TRUNCATE – to remove all records from a table quickly (without logging each row deletion)

DDL changes the schema of the database and is usually auto-committed, meaning changes are permanent immediately.

2. Explain the CREATE command and its syntax.

The CREATE command is used to create new database objects, such as tables or views.

Syntax for creating a table:

```
CREATE TABLE table_name (
    column1 datatype [constraint],
    column2 datatype [constraint],
    ...
);
```

Example:

```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE,
    salary DECIMAL(10,2) CHECK (salary > 0),
    department_id INT
);
```

3. What is the purpose of specifying data types and constraints during table creation?

Data Types:

- Define the kind of data each column can store (e.g., INT, VARCHAR, DATE, DECIMAL).
- Ensure data consistency and optimize storage.
- Prevent invalid data (e.g., storing text in a numeric column).

Constraints:

- Enforce rules at the column or table level.
- Maintain data integrity (e.g., ensuring values are not null, are unique, or follow specific rules).
- Establish relationships between tables (via PRIMARY KEY and FOREIGN KEY).

Specifying data types and constraints up front ensures that the database is robust, reliable, and resistant to data anomalies or corruption.

5. ALTER Command

Theory Questions:

1. What is the use of the ALTER command in SQL?

The ALTER command in SQL is used to modify the structure of an existing database table. It allows you to:

- Add new columns
- Modify existing columns
- Drop (remove) columns
- Rename columns or the table itself
- Add or remove constraints (e.g., PRIMARY KEY, FOREIGN KEY)

2. How can you add, modify, and drop columns using ALTER?

a. Add a Column

ALTER TABLE table_name

ADD column_name datatype;

Example:

ALTER TABLE employees

ADD date_of_birth DATE;

b. Modify a Column

ALTER TABLE table_name

MODIFY column_name new_datatype;

Example (MySQL):

ALTER TABLE employees

MODIFY salary DECIMAL(10,2);

Example (PostgreSQL / SQL Server):

ALTER TABLE employees

ALTER COLUMN salary TYPE DECIMAL(10,2);

c. Drop a Column

ALTER TABLE table_name

DROP COLUMN column_name;

Example:

ALTER TABLE employees

DROP COLUMN middle_name;

6. DROP Command

Theory Questions:

1. What is the function of the DROP command in SQL?

The DROP command in SQL is used to completely remove database objects such as:

- Tables
- Views
- Indexes
- Stored procedures
- Databases

When you use DROP, the object is permanently deleted from the database, along with all of its data and structure.

Example:

DROP TABLE employees;

This command removes the employees table and all the data stored in it.

2. What are the implications of dropping a table from a database?

Dropping a table has significant and irreversible consequences:

Implications:

- Permanent Data Loss: All data in the table is permanently deleted.
- Schema Loss: The table's structure (columns, data types, constraints) is erased.
- Constraint Impacts: Any foreign key relationships to/from other tables will be broken.
- Dependent Objects: Views, stored procedures, triggers, or functions that reference the table may stop working or throw errors.
- No Undo: SQL doesn't have a built-in undo for a DROP operation. Recovery would require restoring from a backup.

7. Data Manipulation Language (DML)

Theory Questions:

1. Define the INSERT, UPDATE, and DELETE commands in SQL

a. INSERT

The INSERT command is used to add new rows of data into a table.

Syntax:

INSERT INTO table_name (column1, column2, ...)

VALUES (value1, value2, ...);

Example:

INSERT INTO employees (name, position, salary)

VALUES ('John Doe', 'Manager', 75000);

b. UPDATE

The UPDATE command is used to modify existing data in a table.

Syntax:

UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

Example:

UPDATE employees

SET salary = 80000

WHERE name = 'John Doe';

c. DELETE

The DELETE command is used to remove existing rows from a table.

Syntax:

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

Example:

```
DELETE FROM employees
WHERE name = 'John Doe';
```

2. What is the importance of the WHERE clause in UPDATE and DELETE operations?

The WHERE clause is crucial in both UPDATE and DELETE statements because it specifies which rows should be affected.

Without a WHERE clause:

- UPDATE will modify all rows in the table.
- DELETE will remove all rows in the table.

Importance:

- Prevents accidental data loss or unwanted updates.
- Targets specific rows based on conditions (e.g., ID, name, status).
- Ensures data integrity and accuracy during operations.

Example without WHERE:

```
DELETE FROM employees;
-- Deletes ALL employees. Often irreversible!
```

Example with WHERE:

```
DELETE FROM employees
WHERE employee_id = 101;
-- Deletes only the employee with ID 101
```

8. Data Query Language (DQL)

Theory Questions:

1. What is the SELECT statement, and how is it used to query data?

The SELECT statement in SQL is used to retrieve data from one or more tables in a database. It allows you to specify:

- Which columns you want
- From which table(s)
- Optional conditions, ordering, grouping, and more

Basic Syntax:

SELECT column1, column2, ...

FROM table_name;

Example:

SELECT name, salary

FROM employees;

This returns the name and salary columns from the employees table.

2. Explain the use of the ORDER BY and WHERE clauses in SQL queries

a. WHERE Clause

The WHERE clause is used to filter rows based on specific conditions. It allows you to return only the rows that meet the given criteria.

Syntax:

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

Example:

```
SELECT name, salary
FROM employees
WHERE salary > 50000;
```

→ Returns only employees with a salary greater than 50,000.

b. ORDER BY Clause

The ORDER BY clause is used to sort the result set by one or more columns, either in ascending (ASC) or descending (DESC) order.

Syntax:

```
SELECT column1, column2
FROM table_name
ORDER BY column1 [ASC|DESC], column2 [ASC|DESC];
```

Example:

SELECT name, salary

FROM employees

ORDER BY salary DESC;

→ Returns employee names and salaries, sorted by salary from highest to lowest.

Combine WHERE and ORDER BY:

SELECT name, salary

FROM employees

WHERE salary > 50000

ORDER BY name ASC;

9. Data Control Language (DCL)

Theory Questions:

1. What is the purpose of GRANT and REVOKE in SQL?

GRANT:

The GRANT command is used to **give specific privileges** (permissions) to users or roles on database objects (like tables, views, or procedures).

REVOKE:

The REVOKE command is used to **take away privileges** that were previously granted to users or roles.

2. How do you manage privileges using these commands?

You use GRANT and REVOKE to control who can perform actions such as:

- SELECT – Read data from a table
- INSERT – Add data to a table
- UPDATE – Modify data in a table
- DELETE – Remove data from a table
- ALL PRIVILEGES – Grant all permissions

a. GRANT Syntax:

GRANT privilege_list

ON object_name

TO user_or_role;

Example:

GRANT SELECT, INSERT

ON employees

TO john;

→ Grants john the ability to read from and insert into the employees table.

b. REVOKE Syntax:

REVOKE privilege_list

ON object_name

FROM user_or_role;

Example:

REVOKE INSERT

ON employees

FROM john;

→ Removes john's ability to insert data into the employees table.

Best Practices for Managing Privileges:

- **Grant only necessary permissions** (principle of least privilege).
- **Use roles** to group users and manage privileges more efficiently.
- **Revoke permissions immediately** when users no longer need access.

10. Transaction Control Language (TCL)

Theory Questions:

1. What is the purpose of the COMMIT and ROLLBACK commands in SQL?

COMMIT:

- The COMMIT command is used to **save all changes** made during the current transaction **permanently** to the database.
- Once committed, the changes **cannot be undone**.

Example:

BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 500 WHERE account_id = 101;

UPDATE accounts SET balance = balance + 500 WHERE account_id = 202;

COMMIT;

→ Transfers money between two accounts and saves the changes.

ROLLBACK:

- The ROLLBACK command is used to **undo all changes** made in the current transaction **since the last COMMIT or BEGIN**.
- Useful when something goes wrong or you want to cancel the operation.

Example:

BEGIN TRANSACTION;

DELETE FROM employees WHERE department = 'HR';

ROLLBACK;

→ Cancels the deletion and restores the data.

2. Explain how transactions are managed in SQL databases

What is a Transaction?

A **transaction** is a sequence of one or more SQL operations that are **executed as a single unit of work**. It must follow the **ACID** properties:

- **A**tomicity – All or nothing
- **C**onsistency – Data must remain valid before and after
- **I**solation – Transactions should not interfere with each other
- **D**urability – Once committed, changes persist even after failure

Managing Transactions in SQL:

Standard flow:

```
BEGIN TRANSACTION;    -- or START TRANSACTION
-- SQL statements (INSERT, UPDATE, DELETE, etc.)
COMMIT;               -- or ROLLBACK if needed
```

Example with error handling:

```
BEGIN TRANSACTION;

UPDATE products SET stock = stock - 1 WHERE product_id = 1001;
UPDATE orders SET status = 'shipped' WHERE order_id = 5002;

-- Suppose something goes wrong here:
-- ROLLBACK;
```

11. SQL Joins

Theory Questions:

1. Explain the concept of JOIN in SQL

A **JOIN** in SQL is used to **combine rows from two or more tables** based on a **related column** between them (usually a foreign key).

This allows you to retrieve meaningful, combined data across tables that are logically connected.

Types of JOINs and Their Differences

🔷 INNER JOIN

- Returns **only the rows** where there is a match in **both** tables.
- Rows without a match are excluded.

Example:

```
SELECT employees.name, departments.department_name
FROM employees
INNER JOIN departments ON employees.department_id = departments.id;
```

LEFT JOIN (or LEFT OUTER JOIN)

- Returns **all rows from the left table**, and matched rows from the right table.
- If no match exists, NULLs are returned for columns from the right table.

Example:

```
SELECT employees.name, departments.department_name
```

FROM employees

LEFT JOIN departments ON employees.department_id = departments.id;

RIGHT JOIN (or RIGHT OUTER JOIN)

- Returns **all rows from the right table**, and matched rows from the left table.
- If no match exists, NULLs are returned for columns from the left table.

Example:

SELECT employees.name, departments.department_name

FROM employees

RIGHT JOIN departments ON employees.department_id = departments.id;

FULL OUTER JOIN

- Returns **all rows from both tables**.
- If there is no match, NULLs will appear for missing columns from either table.

Example:

SELECT employees.name, departments.department_name

FROM employees

FULL OUTER JOIN departments ON employees.department_id = departments.id;

2. How are JOINs used to combine data from multiple tables?

JOINs are used to:

- **Connect related data** stored in different tables (e.g., employees & departments, orders & customers).
- **Query complex relationships** without duplicating data.
- **Create powerful views** of your database by merging data logically.

Example: Combining orders and customers

```
SELECT orders.order_id, customers.name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.id;
```

Summary Table

JOIN Type	Returns Rows From	Includes Unmatched Rows
INNER JOIN	Both tables	✗ No
LEFT JOIN	Left table	✅ Yes (from left)
RIGHT JOIN	Right table	✅ Yes (from right)
FULL JOIN	Both tables	✅ Yes (both sides)

12. SQL Group By

Theory Questions:

1. What is the GROUP BY clause in SQL? How is it used with aggregate functions?

GROUP BY Clause:

The GROUP BY clause is used to **group rows that have the same values** in specified columns into summary rows—often used with **aggregate functions** to perform calculations for each group.

Common Aggregate Functions:

- COUNT() – Number of rows
- SUM() – Total sum
- AVG() – Average value
- MAX() – Maximum value
- MIN() – Minimum value

How it's used:

Syntax:

SELECT column_name, AGGREGATE_FUNCTION(column_name)

FROM table_name

GROUP BY column_name;

Example:

Group employees by department and calculate the average salary per department:

```
SELECT department_id, AVG(salary) AS average_salary
FROM employees
GROUP BY department_id;
```

2. Explain the difference between GROUP BY and ORDER BY

Feature	GROUP BY	ORDER BY
Purpose	Groups rows for aggregation	Sorts rows in the output
Used With	Often used with aggregate functions	Used to organize result display
Output	One row per group	All rows, possibly grouped if sorted
Order?	Does not sort results (unless combined with ORDER BY)	Sorts rows by one or more columns

ORDER BY Example:

Sort employees by salary (descending):

```
SELECT name, salary
FROM employees
ORDER BY salary DESC;
```

GROUP BY + ORDER BY Example:

Group by department, calculate total salary, and sort by total:

```
SELECT department_id, SUM(salary) AS total_salary
FROM employees
GROUP BY department_id
ORDER BY total_salary DESC
```

13. SQL Stored Procedure

Theory Questions:

1. What is a stored procedure in SQL, and how does it differ from a standard SQL query?

Stored Procedure:

A **stored procedure** is a **precompiled set of SQL statements** (and optionally control logic like IFs, loops, parameters, etc.) that is **stored in the database** and can be **executed repeatedly** by name.

Syntax Example (MySQL):

```
CREATE PROCEDURE GetEmployeeByID(IN emp_id INT)
BEGIN
  SELECT * FROM employees WHERE id = emp_id;
END;
```

You can call it like this:

```
CALL GetEmployeeByID(101);
```

How it differs from a standard SQL query:

Feature	Stored Procedure	Standard SQL Query
Definition	Named block of SQL code stored in the database	One-time SQL statement
Reusability	Can be called multiple times	Must be re-entered or re-written each time
Logic Support	Supports loops, conditions, variables	No control flow (just pure SQL)

Parameters	Accepts input/output parameters	Typically hardcoded
Execution Speed	Precompiled, often faster for repeated use	Parsed and executed each time

2. Explain the advantages of using stored procedures

Advantages:

1. **Reusability**
    - Write once, use many times across different applications.
2. **Performance**
    - Precompiled and stored on the server → faster for repeated execution.
3. **Maintainability**
    - Centralizes business logic in the database. Changes made in one place.
4. **Security**
    - Restrict direct access to tables. Grant users permission to call the procedure only.
5. **Reduced Network Traffic**
    - Executes multiple SQL statements in one call, minimizing client-server communication.
6. **Supports Logic and Error Handling**
    - Can include IF, LOOP, CASE, and error control like DECLARE ... HANDLER.
7. **Encapsulation of Business Logic**
    - Keeps business rules inside the database, reducing duplication across applications.

14. SQL View

Theory Questions:

1. What is a view in SQL, and how is it different from a table?

What is a View?

A **view** in SQL is a **virtual table** based on a SELECT query. It **does not store data physically**—instead, it dynamically presents data from one or more tables when queried.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2, ...

FROM table_name

WHERE condition;

Example:

CREATE VIEW high_salary_employees AS

SELECT name, salary

FROM employees

WHERE salary > 70000;

→ Now, querying high_salary_employees works just like a table:

SELECT * FROM high_salary_employees;

Differences: View vs Table

Feature	View	Table
Data Storage	No (virtual)	Yes (physical)
Updatable?	Sometimes (read-only by default)	Yes
Source	Based on a SELECT query from table(s)	Stores actual data
Flexibility	Can simplify complex queries	Holds raw data
Security	Can restrict access to specific columns/rows	Direct access to all data

2. Advantages of Using Views in SQL Databases

1. Simplifies Complex Queries

- Abstracts and hides complexity.
- Example: Join across multiple tables → view → simple query.

2. Enhances Security

- Users can be granted access to views without exposing the underlying tables.
- You can limit access to specific columns or rows.

3. Improves Maintainability

- If business logic changes, you only update the view—not every query that uses it.

4. Promotes Data Consistency

- Centralized business logic in views reduces the chance of logic duplication.

5. Supports Logical Data Independence

- You can change the structure of base tables without affecting applications using views (as long as the view definition stays valid).

6. Read-Only Snapshots

- You can create views to represent specific “slices” of data (e.g., monthly sales) without affecting the original tables.

15. SQL Triggers

Theory Questions:

1. What is a trigger in SQL? Describe its types and when they are used.

What is a Trigger?

A **trigger** is a special kind of stored procedure that **automatically executes (or "fires") in response to certain events** on a table or view, such as when rows are inserted, updated, or deleted.

Triggers are typically used for:

- Enforcing complex business rules
- Auditing changes
- Maintaining derived or summary data
- Validating input data beyond constraints

Types of Triggers:

Triggers can be classified based on **when** and **what** they act upon:

Trigger Type	When It Fires	Description	Use Case Example
BEFORE INSERT	Before a new row is inserted	Modify or validate data before insert	Check if new salary > minimum wage
AFTER INSERT	After a new row is inserted	Log insert operation	Insert audit log entry

BEFORE UPDATE	Before a row is updated	Validate or modify data before update	Ensure new value meets criteria
AFTER UPDATE	After a row is updated	Track changes	Update modification timestamp
BEFORE DELETE	Before a row is deleted	Prevent or log deletion	Prevent deletion if conditions not met
AFTER DELETE	After a row is deleted	Clean up related data	Remove dependent records

2. Explain the difference between INSERT, UPDATE, and DELETE triggers

Trigger Type	Event It Responds To	Typical Use Case
INSERT Trigger	Fires when a new row is added	Validate new data, auto-fill fields, or log new entries
UPDATE Trigger	Fires when existing row(s) change	Enforce business rules on updated data, audit changes, or maintain summary data
DELETE Trigger	Fires when row(s) are removed	Prevent unauthorized deletion, log deleted data, cascade deletes

Example Scenario:

- **INSERT Trigger:** When a new employee is added, automatically set their hire date to the current date.
- **UPDATE Trigger:** When an employee's salary changes, log the old and new salary values.
- **DELETE Trigger:** When an employee is removed, archive their data in another table.

Summary:

Aspect	INSERT Trigger	UPDATE Trigger	DELETE Trigger
When it fires	On new row insertion	On row update	On row deletion
Typical actions	Validate/modify new data	Enforce rules, log changes	Prevent or log deletion
Data access	Access to NEW row data	Access to OLD and NEW row data	Access to OLD row data

16. Introduction to PL/SQL

Theory Question

1. What is PL/SQL, and how does it extend SQL's capabilities?

PL/SQL (Procedural Language/SQL) is Oracle's procedural extension to standard SQL.

- It combines the power of SQL with procedural programming constructs like **loops, conditionals, variables, and exception handling**.
- Allows you to write **blocks of code** (called **anonymous blocks**, **procedures**, **functions**, **packages**, and **triggers**) that can execute complex logic inside the database.
- Extends SQL by enabling **control flow**, **error handling**, and **modularity**, which standard SQL alone cannot provide.

How PL/SQL extends SQL:

Feature	SQL	PL/SQL
Programming Logic	No	Yes (loops, IFs, CASE, etc.)
Variables	No	Yes
Error Handling	Limited	Robust exception handling
Modularity	No	Supports procedures & functions
Control Structures	No	Yes
Transaction Control	Basic	Supports complex transaction logic

2. Benefits of using PL/SQL

1. Improved Performance

- Code runs directly inside the Oracle database, reducing network traffic by executing multiple SQL statements in a single block.
- PL/SQL is **precompiled**, which improves execution speed.

2. Enhanced Productivity

- Supports procedural constructs making it easier to write complex business logic.
- Code can be reused via procedures, functions, and packages.

3. Better Error Handling

- Supports exception handling to catch and manage runtime errors gracefully.

4. Modularity

- Programs can be broken into smaller, manageable, and reusable blocks (procedures, functions, packages).

5. Security

- Business logic encapsulated in the database restricts direct access to underlying tables.
- You can grant execute privileges on procedures/packages without giving direct table access.

6. Portability

- PL/SQL code is portable across different Oracle database versions.

7. Integration

- Seamlessly integrates SQL statements with procedural constructs in the same block.

17. PL/SQL Control Structures

Theory Questions:

1. What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.

Control Structures in PL/SQL

Control structures allow you to control the flow of execution within a PL/SQL program block. They include:

- Conditional statements (e.g., IF-THEN, CASE)
- Looping statements (e.g., LOOP, WHILE, FOR)
- Exception handling

IF-THEN Control Structure

Used to execute a block of code only if a specified condition is true.

Syntax:

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

LOOP Control Structure

Used to repeat a block of statements multiple times.

2. How do control structures in PL/SQL help in writing complex queries?

Benefits of Control Structures:

- **Conditional Logic:** You can apply different logic paths depending on data values, allowing complex decision-making beyond basic SQL filtering.
- **Repetitive Processing:** Loops let you process multiple rows or repeat tasks without writing repetitive code, which is essential when SQL alone can't handle procedural logic.
- **Modularity & Readability:** By using control structures, you organize complex logic clearly, making maintenance easier.
- **Exception Handling:** Control structures allow you to handle errors gracefully and keep data consistent.
- **Dynamic Operations:** You can build logic that adapts to changing data or business rules, something standard SQL queries cannot do.

Summary

Control Structure	Purpose	Example Use Case
IF-THEN	Conditional execution based on logic	Apply discounts based on salary
LOOP	Repeat execution until condition met	Process batch updates on rows
WHILE	Repeat while a condition is true	Read data until end of result set
FOR	Iterate fixed number of times	Generate reports for last 12 months

18. SQL Cursors

Theory Questions:

1. What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.

What is a Cursor?

A **cursor** is a pointer that allows you to **process query result sets row by row** in PL/SQL.

- When you run a SELECT statement that returns multiple rows, a cursor helps you fetch and manipulate each row individually.
- Think of it as a **handle or control structure** to traverse through the result set.

Implicit Cursor

- Automatically created by Oracle whenever you execute a **single SQL statement** like SELECT INTO, INSERT, UPDATE, or DELETE.
- You **do not declare or control** implicit cursors directly.
- Oracle manages the lifecycle of implicit cursors behind the scenes.
- You can access its attributes like %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN to check status.

Explicit Cursor

- You **declare explicitly** when you need to fetch multiple rows **one row at a time**.
- Gives you full control to **open, fetch, and close** the cursor.
- Useful when handling result sets with multiple rows.

2. When would you use an explicit cursor over an implicit one?

Scenario	Use Implicit Cursor	Use Explicit Cursor
Single-row queries	✅ Automatically handled	❌ Overkill
Multi-row queries	❌ Implicit cursors cannot loop through rows	✅ Explicit cursors needed to fetch row by row
Need control over cursor lifecycle	❌ Implicit is automatic	✅ Explicit cursor control (open, fetch, close)
Complex row-by-row processing or logic	❌ Limited	✅ Provides flexibility to handle each row
Performance considerations	✅ Simpler and faster for single-row queries	✅ Allows optimized fetching and processing

Summary:

Aspect	Implicit Cursor	Explicit Cursor
Declaration	No (automatic)	Yes (must declare explicitly)
Usage	Single-row SQL statements	Multi-row query processing
Control	No control over opening/closing	Full control of cursor lifecycle
Row-by-row processing	Not possible	Possible
Syntax complexity	Simple	More complex

19. Rollback and Commit Savepoint

Theory Questions:

1. Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?

SAVEPOINT

- A **SAVEPOINT** is a **marker or a checkpoint within a transaction**.
- It allows you to **set intermediate points** inside a transaction, so you can **roll back partially** to that point without rolling back the entire transaction.
- SAVEPOINTs help manage complex transactions by giving more control over error recovery.

Interaction with ROLLBACK and COMMIT:

- **ROLLBACK TO SAVEPOINT savepoint_name;**
  Rolls back only the part of the transaction after the specified savepoint, keeping the previous work intact.
- **ROLLBACK;**
  Rolls back the **entire transaction**, ignoring savepoints.
- **COMMIT;**
  Commits the **entire transaction**, including all savepoints. Once committed, savepoints are released and cannot be rolled back to.

2. When is it useful to use savepoints in a database transaction?

Use cases for savepoints:

1. **Complex Transactions with Multiple Steps:**
   When a transaction has multiple logical steps, you can save progress and roll back only part of the transaction on error.
2. **Error Handling and Recovery:**
   If an error occurs midway, instead of aborting the entire transaction, you can rollback to the last savepoint and continue.
3. **Partial Undo:**
   When only part of the changes need to be undone while preserving others.
4. **Nested Transaction Simulation:**
   SQL databases don't always support true nested transactions, but savepoints provide a way to mimic this behavior.
5. **Performance Optimization:**
   Avoids the cost of rolling back and redoing the entire transaction by limiting rollback scope.

Summary:

Command	Effect
SAVEPOINT name;	Sets a checkpoint within the transaction
ROLLBACK TO name;	Undo changes after the savepoint
ROLLBACK;	Undo entire transaction
COMMIT;	Finalize entire transaction