Python – Backend Assignment

Module 4 – Introduction to DBMS

1. Introduction to SQL :

❖THEORY EXERCISE:

- **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 perform a variety of database operations, such as querying, updating, inserting, and deleting data.
  - **Why SQL is essential in database management:**
    - ✓ Data Retrieval:
    - ✓ Data Manipulation:
    - ✓ Database Structure Management:
    - ✓ Standardization:
    - ✓ Data Integrity and Security:
    - ✓ Efficiency:

- **Explain the difference between DBMS and RDBMS.**

Feature	DBMS (Database Management System)	RDBMS (Relational DBMS)
Data Structure	Can store data in any format (e.g., hierarchical, network)	Data is stored in tables (relations)
Data Relationships	Does not support relationships between tables	Supports relationships using primary and foreign keys
Normalization	May not support normalization	Supports normalization to reduce redundancy
Examples	XML DB, Hierarchical DB	MySQL, Oracle, PostgreSQL

- **Describe the role of SQL in managing relational databases.**
  - Data Definition:

- Data Manipulation:
- Data Querying:
- Data Integrity and Constraints:
- Transaction Management:
- Security and User Management:
- Database Administration and Optimization:

- **What are the key features of SQL?**
  - Data Querying
  - Data Manipulation
  - Transaction Control
  - Join Operations
  - Subqueries
  - Indexes

❖ Lab Exercises:

- **Create a new database named school_db and a table called students with the following columns: student_id, student_name, age, class, and address.**

Ans:

```
CREATE DATABASE school_db;

CREATE TABLE students

(

   student_id INT,

   student_name VARCHAR(25),

   age INT,

   class INT,

   address TEXT

);
```

- **Insert five records into the students table and retrieve all records using the SELECT statement.**

Ans:

INSERT INTO students VALUES(1, 'Saurya', 18, 12, 'Thaltej'), (2, 'Krish', 19, 12, 'Bodakdev'), (3, 'Yash', 16, 10, 'Gota'), (4 ,'Rudra', 17, 11, 'Bodakdev'), (5, 'Vyom', 18, 12, 'Satadhar');

SELECT * FROM students;

student_id	student_name	age	class	address
1	Kevin	19	12	Modasa
2	Nayan	18	12	Tintoi
3	Het	17	10	Kudol
4	Meet	18	11	Isrol
5	Manthan	19	12	Tintoi

2. SQL Syntax :

❖THEORY EXERCISE:

- **What are the basic components of SQL syntax?**
  - Keywords
  - Identifiers
  - Operators
  - Clauses
  - Expressions
  - Functions
  - Values
  - Comments

- **Write the general structure of an SQL SELECT statement.**
  - **General Structure of an SQL SELECT Statement:**

```
SELECT [columns]
FROM [table_name]
[WHERE condition]
[GROUP BY columns]
[HAVING condition]
[ORDER BY columns]
[LIMIT number_of_records];
```

- **Explain the role of clauses in SQL statements.**
  - **SELECT=** Specifies the columns or expressions to retrieve.
  - **FROM =** Defines the table(s) to query data from.
  - **WHERE =** Filters records based on a condition.
  - **GROUP BY =** Groups rows based on one or more columns for aggregation.
  - **HAVING =** Filters groups after the `GROUP BY` clause.
  - **ORDER BY =** Sorts the result set based on one or more columns.
  - **LIMIT =** Restricts the number of rows returned.
  - **JOIN =** Combines data from multiple tables based on a related column.

❖ Lab Exercises:

- **Write SQL queries to retrieve specific columns (student_name and age) from the students table.**

Ans:

SELECT student_name, age FROM students;

student_name	age
Kevin	19
Nayan	18
Het	17
Meet	18
Manthan	19

- **Write SQL queries to retrieve all students whose age is greater than 10.**

Ans:

SELECT * FROM students WHERE age > 10;

student_id	student_name	age	class	address
1	Kevin	19	12	Modasa
2	Nayan	18	12	Tintoi
3	Het	17	10	Kudol
4	Meet	18	11	Isrol
5	Manthan	19	12	Tintoi

3. SQL Constraints :

❖THEORY EXERCISE:

- What are constraints in SQL? List and explain the different types of constraints.
  - **NOT NULL =** Ensures that a column cannot contain NULL values.
  - **UNIQUE =** Ensures that all values in a column are distinct.
  - **PRIMARY KEY =** Uniquely identifies each record in a table and prevents NULL values.
  - **FOREIGN KEY =** Ensures referential integrity by linking columns across tables.
  - **CHECK =** Enforces a condition that values in a column must satisfy.
  - **DEFAULT =** Provides a default value for a column when no value is specified.
  - **INDEX =** Improves query performance by indexing a column.

- How do PRIMARY KEY and FOREIGN KEY constraints differ?

Aspect	PRIMARY KEY	FOREIGN KEY
Purpose	Uniquely identifies each record in the table.	Ensures referential integrity between two tables.
Uniqueness	The values must be unique.	The values do not need to be unique (can have duplicates).
Nullability	Cannot be NULL.	Can be NULL (optional relationship).
Relationship	Does not establish relationships between tables.	Establishes a relationship between two tables.

- **What is the role of NOT NULL and UNIQUE constraints?**
  - **NOT NULL Constraint :**
    - ✓ **Role**: The NOT NULL constraint ensures that a column cannot contain NULL values. This means that every record in the table must have a valid, non-missing value for this column.
    - ✓ **Purpose**: It's used when a column must always have a value, for example, a user_id or email in a user table. This constraint prevents data from being incomplete by forcing values to be provided during data insertion.

  - **UNIQUE Constraint :**
    - ✓ **Role**: The UNIQUE constraint ensures that all values in a column (or a combination of columns) are distinct across the table. It prevents duplicate values in the specified column(s).
    - ✓ **Purpose**: It's used when you want to ensure that no two rows have the same value in a particular column (such as a username, email, or product code).

❖ Lab Exercise:

- **Create a table teachers with the following columns: teacher_id (Primary Key), teacher_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).**

Ans:

```
CREATE TABLE teachers

(
```

```
    teacher_id INT PRIMARY KEY,
    teacher_name VARCHAR(25) NOT NULL,
    subject TEXT NOT NULL,
    email TEXT UNIQUE
);
```

- **Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.**

Ans:

```
ALTER TABLE students ADD (teacher_id int, FOREIGN KEY(teacher_id) REFERENCES teachers(teacher_id));
```

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

❖THEORY EXERCISE:

- **Define the SQL Data Definition Language (DDL).**
  - **SQL Data Definition Language (DDL)** is a subset of SQL (Structured Query Language) used to define, modify, and manage database structures like tables, schemas, indexes, and views. DDL statements primarily deal with the **structure** of the database rather than the data itself.
  - **Common DDL Commands:**
    - ✓ Create
    - ✓ Alter
    - ✓ Drop
    - ✓ Rename

- **Explain the CREATE command and its syntax.**
  - The **CREATE** command in SQL is used to define new database objects such as tables, views, indexes, schemas, and databases. It is part of SQL's **Data Definition Language (DDL)** and allows you to structure and organize data in a database.
  - **CREATE DATABASE Syntax** : CREATE DATABASE database_name;
  - **CREATE TABLE Syntax :**
    - ✓ CREATE TABLE table_name (
      column1 datatype [constraint],
      column2 datatype [constraint],

...
);

- **CREATE INDEX Syntax :** CREATE INDEX index_name ON table_name (column_name);

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

- **Purpose of Specifying Data Types**:
  - ✓ **Data Integrity**: Ensures that only valid data can be stored in a column. For example, if a column is defined as INT, only integer values can be inserted. If it is defined as VARCHAR(50), only strings up to 50 characters are allowed.
  - ✓ **Efficient Storage**: Different data types require different amounts of storage. By specifying the correct data type, the database can optimize memory usage.
  - ✓ **Validation**: By specifying a data type, the database can automatically reject invalid data (like trying to insert a string into an INT column). This prevents errors and ensures that the data is accurate.
- **Purpose of Specifying Constraints**:
  - ✓ **Ensuring Data Integrity**: Constraints help enforce rules that the data must follow, ensuring the data's validity. For example, if you specify a NOT NULL constraint, the database ensures that a column must always have a value.
  - ✓ **Enforcing Uniqueness**: Constraints like PRIMARY KEY and UNIQUE ensure that values in a column or combination of columns are unique, preventing duplicate data. This is especially important for identifiers like employee IDs or email addresses.

❖ Lab Exercise:

▪ Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.

Ans:

```
CREATE TABLE courses (
   course_id int,
   course_name text,
   course_credits int,
   PRIMARY KEY (course_id)
);
```

❖ **Use the CREATE command to create a database university_db.**

Ans:

CREATE DATABASE university_db;

5. ALTER Command :

❖THEORY EXERCISE:

- **What is the use of the ALTER command in SQL?**
  - **Common Uses of the ALTER Command:**
    - **Adding a New Column to a Table:**
    - ✓ You can add new columns to an existing table using the ALTER TABLE statement with the ADD keyword.
    - **Modifying an Existing Column :**
    - ✓ You can change the data type or constraints of an existing column.
    - **Renaming a Column :**
    - ✓ You can rename an existing column in a table using ALTER TABLE with the RENAME COLUMN clause (Note: the syntax can vary slightly depending on the database system).
    - **Dropping a Column from a Table :**
    - ✓ You can remove a column from a table using the DROP COLUMN clause.
    - **Renaming a Table :**
    - ✓ You can rename an existing table using the RENAME TO clause.

- **How can you add, modify, and drop columns from a table using ALTER?**
  - **Adding a Column :**
    - To add a new column to an existing table, you use the ADD clause with the **ALTER TABLE** statement. You also specify the column name, its data type, and optionally, any constraints (like NOT NULL or UNIQUE).
    - **Syntax**: ALTER TABLE table_name ADD column_name data_type [constraints];
  - **Modifying an Existing Column :**
    - To modify an existing column (such as changing its data type or adding constraints), you use the MODIFY (or ALTER COLUMN in some databases) clause. The syntax depends on the specific database system.
    - Syntax : ALTER TABLE table_name MODIFY column_name new_data_type [constraints];
  - **Dropping a Column :**

- To remove a column from a table, you use the DROP COLUMN clause. When you drop a column, the data stored in that column will be permanently removed, so this operation should be used carefully.
- Syntax : ALTER TABLE table_name DROP COLUMN column_name;

❖Lab Exercises:

- **Modify the courses table by adding a column course_duration using the ALTER command.**

  Ans:

  ALTER TABLE courses ADD (course_duration int);

- **Drop the course_credits column from the courses table.**

  Ans:

  ALTER TABLE courses DROP course_credits;

6. DROP Command :

❖THEORY EXERCISE:

- **What is the function of the DROP command in SQL?**
  - **Main Uses of the DROP Command:**
  - **Dropping a Table**:
    - The **DROP TABLE** command deletes an entire table and its data from the database.
    - Syntax : DROP TABLE table_name;
  - **Dropping a Column :**
    - Some databases allow you to drop a column from a table using the **DROP COLUMN** command.
    - Syntax : ALTER TABLE table_name DROP COLUMN column_name;
  - **Dropping an Index**:

- The **DROP INDEX** command removes an index from a table. This is typically done to improve performance or to remove unnecessary indexes.
- Syntax : DROP INDEX index_name;

- **Dropping a View :**

- The **DROP VIEW** command deletes a view from the database. A view is a virtual table, and dropping it will remove the virtual representation but not the underlying data.
- Syntax : DROP VIEW view_name;

- **What are the implications of dropping a table from a database?**
  - Loss of Data.
  - Loss of Table Structure.
  - Impact on Dependent Objects.
  - Storage Reclamation.
  - Security Implications.
  - Increased Load on Transaction Log (for Some Systems).

❖ Lab Exercise:

- **Drop the teachers table from the school_db database.**

Ans:

DROP TABLE teachers;

- **2) Drop the students table from the school_db database and verify that the table has been removed.**

Ans:

DROP TABLE students;

SHOW tables;

Tables_in_school_db
courses

7. Data Manipulation Language (DML) :

❖THEORY EXERCISE:

- **Define the INSERT, UPDATE, and DELETE commands in SQL.**
  - **INSERT Command :**
  - The INSERT command is used to add new rows of data into a table. You can insert data into specific columns or all columns in a table.
  - **Syntax :**
  - ✓ INSERT INTO table_name (column1,column2, column3, ...)
    VALUES (value1, value2, value3, ...);
  - **UPDATE Command :**
  - The **UPDATE** command is used to modify existing data in a table. You can update one or more columns for a specific row (or rows) based on a condition specified by the WHERE clause.
  - Syntax :
  - ✓ UPDATE table_nameSET column1 = value1, column2 = value2, ...
    WHERE condition;
  - **DELETE Command :**
  - The **DELETE** command is used to remove rows from a table based on a specified condition. If you omit the WHERE clause, all rows in the table will be deleted, but the table itself remains.
  - Syntax :
  - ✓ DELETE FROM table_name
    WHERE condition;

- **What is the importance of the WHERE clause in UPDATE and DELETE operations?**
  - **Importance of WHERE Clause in UPDATE Operation :**
  - With WHERE Clause :
  - ✓ Updates specific rows based on the condition provided.
  - Without WHERE Clause :
  - ✓ Updates **all rows** in the table
  - **Importance of WHERE Clause in DELETE Operation**
  - With WHERE Clause :
  - ✓ Deletes specific rows based on the condition provided
  - Without WHERE Clause :
  - ✓ Deletes **all rows** in the table.

❖ Lab Exercises:

- **Insert three records into the courses table using the INSERT command.**

  Ans:

  INSERT INTO courses VALUES(1, 'Python', 4), (2, 'Java', 5), (3, 'PHP', 6);

- **2) Update the course duration of a specific course using the UPDATE command.**

  Ans:

  UPDATE courses SET course_duration = 3 WHERE course_name = 'PHP';

- **3) Delete a course with a specific course_id from the courses table using the DELETE command.**

  Ans:

  DELETE FROM courses WHERE course_id = 3;

8. Data Query Language (DQL) :

❖THEORY EXERCISE:

- **What is the SELECT statement, and how is it used to query data?**
  - A "SELECT" statement in SQL (Structured Query Language) is a command used to retrieve specific data from a database table, allowing you to query and extract information based on chosen criteria, effectively pulling out the desired columns and rows from a table to view as a result set.

- **Explain the use of the ORDER BY and WHERE clauses in SQL queries**

  - **WHERE Clause:**

✓ The WHERE clause is used to filter records based on a specified condition. It determines which rows are selected and returned in the query.

✓ You can use operators like =, !=, >, <, >=, <=, BETWEEN, IN, LIKE, IS NULL, and logical operators like AND, OR to create conditions.

  - **ORDER BY Clause**:

✓ The ORDER BY clause is used to sort the result set based on one or more columns. By default, the sorting is in ascending order (ASC), but you can specify descending order (DESC) if needed.

✓ You can sort data by one or more columns, and it can be used with ASC (ascending) or DESC (descending) to control the direction of sorting.

❖ Lab Exercises:

- **Retrieve all courses from the courses table using the SELECT statement.**

Ans:

SELECT * FROM courses;

course_id	course_name	course_duration
1	Python	4
2	Java	5

- **2) Sort the courses based on course_duration in descending order using ORDER BY.**

Ans:

SELECT * FROM courses ORDER BY course_duration DESC;

course_id	course_name	course_duration ▼ 1
2	Java	5
1	Python	4

- **3) Limit the results of the SELECT query to show only the top two courses using LIMIT.**

Ans:

SELECT * FROM courses LIMIT 2;

course_id	course_name	course_duration
1	Python	4
2	Java	5

9. Data Control Language (DCL) :

❖THEORY EXERCISE:

- **What is the purpose of GRANT and REVOKE in SQL?**
  - **Purpose of GRANT and REVOKE:**
  - **GRANT**: It allows the database administrator or authorized user to give permissions to other users or roles to access and perform operations on database objects.
  - **REVOKE**: It allows the database administrator or authorized user to remove those permissions when they are no longer needed or if access needs to be restricted.

- **How do you manage privileges using these commands?**
  - **Granting Privileges** (Using GRANT):

- **Granting Permissions**: You can grant permissions to users or roles for specific database objects (like tables, views, stored procedures).
- **Multiple Privileges**: You can grant multiple privileges at once

- **Revoking Privileges** (Using REVOKE):

- **Revoking Permissions**: The REVOKE command is used to remove previously granted privileges from users or roles.

- **Managing Privileges with Roles**:

❖ Lab Exercises:

- **Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.**

  Ans:

  CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';

  CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';

  GRANT SELECT ON school.courses TO 'user1'@'localhost';

- **2) Revoke the INSERT permission from user1 and give it to user2.**

  Ans:

  REVOKE INSERT ON your_database.courses FROM 'user1'@'localhost';

  GRANT INSERT ON your_database.courses TO 'user2'@'localhost';

10. Transaction Control Language (TCL) :

❖THEORY EXERCISE:

- **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 to the database permanently.

- ✓ It marks the end of a successful transaction and makes the changes visible to other users or sessions.

- **ROLLBACK**:
- ✓ The ROLLBACK command is used to **undo** the changes made in the current transaction and **restore** the database to its state before the transaction began. It is often used when an error occurs, and you want to discard all changes made during the transaction.
- ✓ A ROLLBACK can only affect the current transaction and will not affect previous transactions

- **Explain how transactions are managed in SQL databases.**
  - Transaction Properties (ACID)
  - Managing Transactions in SQL
  - Isolation Levels in SQL
  - Transaction Management Workflow

❖ Lab Exercises:

- **Insert a few rows into the courses table and use COMMIT to save the changes.**

Ans:

```
START TRANSACTION;

INSERT INTO courses VALUES(3, 'PHP', 6), (4, '.NET', 7);

COMMIT;
```

- **2) Insert additional rows, then use ROLLBACK to undo the last insert operation.**

Ans:

```
START TRANSACTION;

INSERT INTO courses VALUES

(5, 'Machine Learning', 4),

(6, 'Cloud Computing', 6);
```

```
ROLLBACK;
```

- **3) Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.**

Ans:

```
START TRANSACTION;

SAVEPOINT before_changes;

INSERT INTO courses VALUES(3, 'PHP', 6), (4, '.NET', 7);

UPDATE courses SET course_duration = 10 WHERE course_id = 5;

ROLLBACK TO SAVEPOINT before_changes;
```

11. SQL Joins :

❖THEORY EXERCISE:

- **Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?**
    - **Concept of JOIN in SQL :** In SQL, a **JOIN** is used to combine rows from two or more tables based on a related column between them. When you have multiple tables in a database that are linked by keys (usually primary and foreign keys), you use **JOIN** to retrieve data from more than one table in a single query.
    - **INNER JOIN :** Returns only the rows that have matching values in both tables.
    - **LEFT JOIN :** Returns all rows from the left table and matching rows from the right table; non-matching rows from the right table are replaced with NULL
    - **RIGHT JOIN :** Returns all rows from the right table and matching rows from the left table; non-matching rows from the left table are replaced with NULL
    - **FULL OUTER JOIN :** Returns all rows from both tables; non-matching rows from both tables are replaced with NULL.

- **How are joins used to combine data from multiple tables?**
    - Understanding Relationships Between Tables:
    - How Joins Combine Data from Multiple Tables:

- Using Joins with More than Two Tables:
- How to Choose the Right Type of Join:

Lab Exercises:

- **Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.**

Ans:

```
CREATE TABLE departments(
   did INT PRIMARY KEY,
   dname TEXT NOT NULL
);

CREATE TABLE employees(
   eid INT PRIMARY KEY,
   ename TEXT,
   did INT,
   FOREIGN KEY(did) REFERENCES departments(did)
);

INSERT INTO departments VALUES(1, 'Purchase'), (2, 'Sales'), (3, 'Finance'), (4, 'Marketing'), (5, 'Legal');

INSERT INTO employees VALUES (1, 'Saurya', 4), (2, 'Krish', 4), (3, 'Harshil', 2), (4, 'Jinay', 1);
```

SELECT employees.eid, employees.ename, departments.dname FROM employees JOIN departments ON employees.did = departments.did;

eid	ename	dname
1	Kevin	Sales
2	Het	Purchase
3	Manthan	Marketing
4	Sahil	Purchase

- **2) Use a LEFT JOIN to show all departments, even those without employees.**

Ans:

SELECT employees.ename, departments.dname FROM departments LEFT JOIN employees ON employees.did = departments.did;

eid	ename	dname
1	Kevin	Sales
2	Het	Purchase
3	Manthan	Marketing
4	Sahil	Purchase
NULL	*NULL*	Marketing
NULL	*NULL*	Sales

12. SQL Group By :

❖THEORY EXERCISE:

- **What is the GROUP BY clause in SQL? How is it used with aggregate functions?**
  - **GROUP BY Clause in SQL :** The GROUP BY clause in SQL is used to group rows that have the same values in specified columns into summary rows, such as finding the sum, average, count, maximum, or minimum of a particular field. It is typically used with **aggregate functions** to perform calculations on each group of rows.
  - **GROUP BY Works with Aggregate Functions :**

- ✓ COUNT() with GROUP BY
- ✓ SUM() with GROUP BY
- ✓ AVG() with GROUP BY
- ✓ MIN() and MAX() with GROUP BY

- **Explain the difference between GROUP BY and ORDER BY.**

Feature	GROUP BY	ORDER BY
Purpose	To group rows by one or more columns to perform aggregation on each group.	To sort the result set in ascending or descending order.
Used with	Often used with aggregate functions like COUNT(), SUM(), AVG(), etc.	Used to sort the result set based on one or more columns.
Functionality	Groups data into summary rows based on column values.	Sorts rows based on column values (either ascending or descending).
Example Use Case	Calculating the total sales for each product category.	Sorting employees by their salary in descending order.

❖ Lab Exercise:

1) Group employees by department and count the number of employees in each department using GROUP BY.

Ans:

SELECT COUNT(employees.eid) AS "Total Employees", departments.dname FROM employees JOIN departments on employees.did = departments.did GROUP BY departments.dname;

Total Employees	dname
2	Marketing
1	Purchase
1	Sales

2) Use the AVG aggregate function to find the average salary of employees in each department.

Ans:

SELECT COUNT(employees.eid) AS "Total Employees", AVG(salary) as "Average Salary", departments.dname FROM employees JOIN departments on employees.did = departments.did GROUP BY departments.dname;

Total Employees	Average Salary	dname
2	25000.0000	Marketing
1	25000.0000	Purchase
1	35000.0000	Sales

13. SQL Stored Procedure :

❖THEORY EXERCISE:

- **What is a stored procedure in SQL, and how does it differ from a standard SQL query?**
  - **Stored Procedure in SQL :** A stored procedure in SQL is a precompiled collection of one or more SQL statements that can be executed as a single unit.

Feature	Stored Procedure	Standard SQL Query
Definition	A **precompiled** set of SQL statements stored in the database and executed as a unit.	A **single SQL statement** that is executed once.
Execution	Stored procedures are executed by calling the procedure name.	Standard queries are executed directly when written.
Reusability	Can be reused multiple times without rewriting the SQL code.	Typically written and executed on the fly each time.
Security	Provides better security as you can restrict access to the procedure rather than the underlying data.	Direct access to the database objects can be required, which can be a security risk.

- **Explain the advantages of using stored procedures.**

- **Performance**: Stored procedures improve performance by reducing network traffic and allowing for precompilation.
- **Reusability**: You can reuse stored procedures across applications and queries, ensuring consistent logic execution.
- **Security**: They improve security by controlling access to data and reducing the risk of SQL injection.
- **Maintainability**: They allow for centralized logic that simplifies maintenance and updates.
- **Error Handling**: They provide advanced error handling and transaction control, ensuring data integrity.
- **Reduced Client-Side Logic**: They offload processing to the database server, simplifying application logic.

❖ Lab Exercise:

1) Write a stored procedure to retrieve all employees from the employees table based on department.

Ans:

```
DELIMITER $$

CREATE PROCEDURE find_emp(dep_id int)

BEGIN

SELECT employees.eid, employees.ename, employees.salary, departments.did FROM employees JOIN departments on employees.did = departments.did HAVING did = dep_id;

END;

CALL find_emp(1);
```

2) Write a stored procedure that accepts course_id as input and returns the course details.

Ans:

```
DELIMITER $$
```

```
CREATE PROCEDURE get_course(id int)

BEGIN

        SELECT * FROM courses WHERE course_id = id;

end;

CALL get_course(1);
```

course_id	course_name	course_duration
1	Python	4

14. SQL View :

❖THEORY EXERCISE:

- **What is a view in SQL, and how is it different from a table?**
  - **What is a View in SQL? :** A view in SQL is a virtual table that represents the result of a SELECT query. It is a stored query that can be treated like a table but does not physically store the data. Instead, it dynamically retrieves data from one or more tables whenever it is accessed.

Feature	Table	View
Definition	A table is a **physical** object that stores data in rows and columns.	A view is a **virtual** table that stores a **SELECT query** definition but not actual data.
Structure	Tables have a fixed structure, defined by columns with specific data types.	A view's structure is defined by the **SELECT query**; it can include joins, filters, and transformations.
Data Modification	Data in a table can be inserted, updated, or deleted directly.	Views are typically **read-only**, although some views (if updatable) allow modifications.

Indexes	Tables can have indexes to speed up data retrieval.	Views **cannot have indexes**; they rely on indexes in the underlying tables.

- **Explain the advantages of using views in SQL databases.**
  - **Simplify Complex Queries**: Abstract complex logic and reduce the need for users to write intricate SQL.
  - **Data Abstraction and Security**: Hide the underlying complexity and restrict access to sensitive data.
  - **Reusability**: Reuse common queries, ensuring consistency and reducing redundancy.
  - **Improved Security**: Control access to specific data without granting full access to the underlying tables.
  - **Data Consistency**: Provide a consistent and uniform data representation across applications.
  - **Better Organization**: Logical separation and simplified maintenance of queries in the database.

❖ Lab Exercise:

1) Create a view to show all employees along with their department names.

Ans:

CREATE VIEW emp_deparments AS

SELECT employees.eid, employees.ename, employees.salary, departments.dname FROM employees JOIN departments on employees.did = departments.did;

SELECT * FROM emp_deparments;

eid	ename	salary	dname
1	Saurya	20000	Marketing
2	Krish	30000	Marketing
3	Harshil	35000	Sales
4	Jinay	25000	Purchase

2) Modify the view to exclude employees whose salaries are below $50,000.

Ans:

CREATE OR REPLACE VIEW emp_deparments AS

SELECT employees.eid, employees.ename, employees.salary, departments.dname FROM employees JOIN departments on employees.did = departments.did

WHERE employees.salary < 50000;

15. SQL Triggers :

❖THEORY EXERCISE:

- **What is a trigger in SQL? Describe its types and when they are used.**
  - **What is a Trigger in SQL? :** A **trigger** in SQL is **a** special kind of stored procedure that is automatically executed or fired by the database in response to a specific event or action on a particular table or view.
  - **Types of Triggers in SQL:**
    - BEFORE Triggers.
    - AFTER Triggers.
    - INSTEAD OF Triggers.
  - **When Are Triggers Used?**
    - Data Validation.
    - Enforcing Business Rules.

- Maintaining Referential Integrity.
- Auditing and Logging.

- **Explain the difference between INSERT, UPDATE, and DELETE triggers.**

Type of Trigger	When It Is Fired	Common Use Cases	Key Points
INSERT Trigger	Fired **after** or **before** an **INSERT** operation is executed.	Data validation (before data is inserted)	Can be used for validation or modification of new data before insertion.
UPDATE Trigger	Fired **after** or **before** an **UPDATE** operation is executed.	Tracking changes (audit logs) - Preventing certain	Useful for tracking changes or enforcing business rules for updates.
DELETE Trigger	Fired **after** or **before** a **DELETE** operation is executed.	Cascading deletions in related tables	Useful for maintaining referential integrity or tracking deletions.

❖ Lab Exercise:

1) Create a trigger to automatically log changes to the employees table when a new employee is added.

Ans:

```
CREATE TABLE employees_history(
   dep_id int,
   emp_id int,
   name text,
   salary int,
```

```
   time_changed timestamp,
   action_performed text
);
DELIMITER $$
CREATE TRIGGER insert_trigger AFTER INSERT ON employees FOR EACH ROW
BEGIN
        INSERT INTO employees_history(dep_id, emp_id, name, salary, action_performed)
VALUES(new.did, new.eid, new.ename, new.salary, 'Record Inserted');
END;
```

2) Create a trigger to update the last_modified timestamp whenever an employee record in updated.

Ans:

```
CREATE TABLE emp_update_history(
eidint,
enmae text,
salaryint,
last_modified timestamp,
didint
);
DELIMITER $$
CREATE TRIGGER update_trig AFTER UPDATE ON employees FOR EACH ROW
BEGIN
        INSERT INTO emp_update_history(eid, ename, salary, did) VALUES(new.eid,
new.ename, new.salary, new.did);
END;
```

16. Introduction to PL/SQL :

❖THEORY EXERCISE:

- **What is PL/SQL, and how does it extend SQL's capabilities?**
  - **What is PL/SQL? : PL/SQL** (Procedural Language/SQL) is an **extension of SQL** developed by Oracle for managing and manipulating data in Oracle databases.
  - **How PL/SQL Extends SQL's Capabilities :**
    - Procedural Programming Constructs.
    - Exception Handling.
    - Modular Programming.
    - Caching and Performance Optimization.
    - Triggers.
    - Enhanced Security.

- **List and explain the benefits of using PL/SQL.**
  - Integration with SQL : Combines SQL with procedural programming for more powerful data handling.
  - Improved Performance : Bulk operations, reduced round trips to the database, and better caching.
  - Modularity and Reusability : Code can be organized into reusable procedures, functions, and packages.
  - Exception Handling : Catch and handle errors gracefully, ensuring more robust applications.
  - Security : Restrict access to sensitive data and logic through encapsulation.

❖ Lab Exercises:

1) Write a PL/SQL block to print the total number of employees from the employees table.

Ans:

```
DECLARE
        Employees_total NUMBER;
BEGIN
SELECT COUNT(*) INTO employees_total FROM employees;
DBMS_OUTPUT.PUT_LINE('Total Number of Employees: ' || employees_total);
END;
```

2) Create a PL/SQL block that calculates the total sales from an orders table.

Ans:

```
DECLARE
  total_sales NUMBER;
BEGIN
SELECT SUM(order_amount) INTO total_sales FROM orders;
  IF v_total_sales IS NULL THEN
    v_total_sales := 0;
  END IF;
  DBMS_OUTPUT.PUT_LINE('Total Sales: ' || total_sales);
END;
```

17. PL/SQL Control Structures :

❖THEORY EXERCISE:

- **What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.**
  - **Control Structures in PL/SQL :** Control structures in PL/SQL allow you to dictate the flow of execution of your program based on certain conditions or to repeat certain actions.
  - **IF-THEN Control Structure :**
    - The **IF-THEN** structure is used for conditional branching in PL/SQL. It evaluates an expression (a condition) and, based on whether the condition is true or false, it either executes or skips a block of code
  - **LOOP Control Structure :**
    - The **LOOP** structure in PL/SQL is used for executing a set of statements repeatedly until a specific condition is met. PL/SQL provides different

types of loop constructs, such as the **simple `LOOP`**, **`WHILE` loop**, and **`FOR` loop**.

- **How do control structures in PL/SQL help in writing complex queries?**
  - Handling Conditional Logic (IF-THEN, IF-THEN-ELSE)
  - Looping through Data (LOOP, FOR LOOP, WHILE LOOP)
  - Complex Data Validations
  - Error Handling (EXCEPTION Block)
  - Complex Iterations and Nested Loops
  - Reducing the Number of SQL Queries

❖ Lab Exercises:

1) Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

Ans:

```
DECLARE
   v_employee_id   NUMBER := 101;
   v_department_id NUMBER;
BEGIN
   SELECT department_id INTO v_department_id
   FROM employees
   WHERE employee_id = v_employee_id;

   IF v_department_id = 10 THEN
      DBMS_OUTPUT.PUT_LINE('Employee ' || v_employee_id || ' works in the HR department.');
   ELSIF v_department_id = 20 THEN
```

```
    DBMS_OUTPUT.PUT_LINE('Employee ' || v_employee_id || ' works in the Sales
department.');

  ELSE

    DBMS_OUTPUT.PUT_LINE('Employee ' || v_employee_id || ' works in another
department.');

  END IF;

END;
```

2) Use a FOR LOOP to iterate through employee records and display their names.

Ans:

```
DECLARE

  CURSOR emp_cursor IS

    SELECT employee_id, first_name, last_name FROM employees;

BEGIN

  FOR emp_rec IN emp_cursor LOOP

    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_rec.employee_id ||

                ', Name: ' || emp_rec.first_name || ' ' || emp_rec.last_name);

  END LOOP;

END;
```

18. SQL Cursors :

❖THEORY EXERCISE :

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

Feature	Implicit Cursor	Explicit Cursor
Definition	Automatically created by Oracle for single SQL statements.	Explicitly declared and managed by the developer.
Use Case	For simple SQL operations like SELECT INTO, INSERT, UPDATE, DELETE.	For complex SQL queries that return multiple rows.
Cursor Management	Managed automatically by Oracle; no need to open, fetch, or close.	Developer must open, fetch, and close manually.
Performance	Generally faster for single-row operations.	More overhead, but essential for processing multiple rows with complex logic.

- **When would you use an explicit cursor over an implicit one?**
  - You would typically use an **explicit cursor** over an **implicit cursor** in the following situations:
    - ✓ When Processing Multiple Rows.
    - ✓ When You Need to Fetch Rows One by One.
    - ✓ When Handling Large Result Sets.
    - ✓ When You Need to Reuse the Cursor.
    - ✓ When You Need Full Control Over Cursor Behavior.

❖ Lab Exercise:

1) Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

Ans:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, first_name, last_name, department_id, salary
```

```
    FROM employees;

  emp_rec emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;

  LOOP
    FETCH emp_cursor INTO emp_rec;
    EXIT WHEN emp_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_rec.employee_id ||
                ', Name: ' || emp_rec.first_name || ' ' || emp_rec.last_name ||
                ', Department ID: ' || emp_rec.department_id ||
                ', Salary: ' || emp_rec.salary);
  END LOOP;
  CLOSE emp_cursor;
END;
```

2) Create a cursor to retrieve all courses and display them one by one.

Ans:

```
DECLARE
  CURSOR course_cursor IS
    SELECT course_id, course_name, instructor FROM courses;

  course_rec course_cursor%ROWTYPE;
```

```
BEGIN
   OPEN course_cursor;

   LOOP
      FETCH course_cursor INTO course_rec;
      EXIT WHEN course_cursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('Course ID: ' || course_rec.course_id ||
                  ', Course Name: ' || course_rec.course_name ||
                  ', Instructor: ' || course_rec.instructor);
   END LOOP;

   CLOSE course_cursor;
END;
```

19. Rollback and Commit Savepoint :

❖THEORY EXERCISE:

- **Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?**
  - **Concept of SAVEPOINT in Transaction Management** : In SQL, a SAVEPOINT is a marker that allows you to define a point within a transaction to which you can later ROLLBACK if needed, without rolling back the entire transaction. It is useful for partial rollbacks where you want to undo certain changes in a transaction but not all of them.
  - **How SAVEPOINT Works:**
    - You can set a SAVEPOINT at any point in a transaction.
    - You can then execute further SQL commands or changes.

- If a problem arises or you decide to undo the work done after the savepoint, you can ROLLBACK to that specific savepoint, which will undo all changes made after it but keep the changes made before it.
- If everything is fine and you want to keep all changes, you can COMMIT the entire transaction, including the changes made before and after the savepoint.

- **When is it useful to use savepoints in a database transaction?**
  - **Complex transactions** involving multiple steps or operations that may fail at different points.
  - **Error handling** where you want to undo only specific parts of a transaction while keeping other successful operations intact.
  - **Conditional rollbacks** based on business logic or validation criteria during the transaction.
  - **Long-running transactions** with multiple operations that could benefit from partial rollback if something goes wrong.
  - **Nested transactions**, where you want to simulate the rollback of individual components within a transaction.

❖ Lab Exercises:

1) Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

Ans:

```
INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)

VALUES (5001, 'John', 'Doe', 10, 50000);


SAVEPOINT before_insertion;


INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)

VALUES (5002, 'Jane', 'Smith', 20, 60000);
```

```
INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)
VALUES (5003, 'Alice', 'Johnson', 30, 55000);

ROLLBACK TO before_insertion;
COMMIT;
```

2) Commit part of a transaction after using a savepoint and then rollback the remaining changes.

Ans:

```
INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)
VALUES (6001, 'Tom', 'Anderson', 10, 55000);

SAVEPOINT before_more_inserts;

INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)
VALUES (6002, 'Emma', 'Brown', 20, 60000);

INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)
VALUES (6003, 'Liam', 'Wilson', 30, 65000);

COMMIT;
ROLLBACK TO before_more_inserts;
```