Assignment

Database

Introduction to SQL

Theory Questions:

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

Ans. Structured Query Language (SQL) refers to a standard programming language utilized to extract, organize, manage, and manipulate data stored in relational databases.

SQL plays a essential role in retrieving relevant data from databases, which can later be used by various platforms such as Python or R for analysis purposes. SQL can manage several data transactions simultaneously where large volumes of data are written concurrently.

2. Explain the difference between DBMS and RDBMS.

Ans.

|  |  |
| --- | --- |
| DBMS | RDBMS |
| DBMS stores data as file. | RDBMS stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |
| Normalization is not present. | Normalization is present. |
| DBMS does not support distributed database. | RDBMS supports distributed database. |

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

Ans. SQL (Structured Query Language) plays a fundamental role in managing relational databases. It is the standard programming language specifically designed for interacting with and manipulating relational database systems.

Functions of SQL:

1. Data Definition (DDL)

SQL provides commands to define the structure of a database, including creating, altering, and deleting tables, schemas, and other database objects. Examples:

CREATE TABLE for creating new tables.

ALTER TABLE for modifying existing table structures.

DROP TABLE for deleting tables.

2. Data Manipulation (DML)

SQL allows users to insert, update, delete, and retrieve data within the database. Examples:

INSERT INTO to add new records.

UPDATE to modify existing records.

DELETE to remove records.

SELECT to retrieve data, often combined with filters (WHERE), grouping (GROUP BY), sorting (ORDER BY), and aggregation functions.

3. Data Control (DCL)

SQL includes commands to manage user access and permissions:

GRANT to provide specific privileges to users or roles.

REVOKE to remove those privileges.

4. Transaction Control (TCL)

SQL helps ensure data integrity and consistency through transaction management:

BEGIN TRANSACTION, COMMIT, and ROLLBACK are used to control transactions, ensuring atomicity and recoverability.

4. What are the key features of SQL?

Ans.

1. Easy to Learn and Use: SQL is a declarative language, meaning you specify what you want to do, and the database system handles how to do it.Its syntax is simple and intuitive, making it accessible even for non-programmers.

2. Data Definition Language (DDL):SQL provides commands to define the structure of a database:

CREATE to create tables, views, and other database objects.

ALTER to modify existing objects.

DROP to delete database objects.

3. Data Manipulation Language (DML): It allows users to perform operations on the data stored in the database:

INSERT to add records.

UPDATE to modify existing records.

DELETE to remove records.

SELECT to retrieve data.

LAB EXERCISES:

Lab 1: 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. Database: CREATE DATABASE school\_db;

Table: CREATE TABLE students(

student\_id int PRIMARY KEY AUTO\_INCREMENT,

student\_name varchar (80),

student\_age bigint UNIQUE KEY,

student\_class bigint UNIQUE KEY,

student\_address varchar (80)

);

Lab 2: Insert five records into the students table and retrieve all records using the SELECT statement.

Ans. INSERT INTO students(student\_id,student\_name,student\_age,student\_class,student\_address) VALUES (1,"Aditi",15,9,"Vastral"),

(2,"Barkha",14,8,"Naroda"),

(3,"Kalpesh",16,10,"Maninagar"),

(4,"Aman",17,11,"Khokra"),

(5,"Kevin",18,12,"Sola");

2. SQL Syntax

Theory Questions:

1. What are the basic components of SQL syntax?

Ans. The basic syntax is as follows: SELECT columnName FROM yourTable WHERE CONTAINS ( columnName, 'yourSubstring' );

From the example above, you should note that the column name isn't enclosed in quotes, but the arguments for the CONTAINS SQL function are enclosed in parentheses.

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

Ans. SELECT column1, column2, ...

FROM table\_name;

3. Explain the role of clauses in SQL statements. Ans.

1. SELECT – Specifies the columns to retrieve from a table.

Eg: SELECT name, age FROM employees;

2. FROM – Identifies the table(s) from which data is fetched.

Eg: SELECT \* FROM employees;

3. WHERE – Filters records based on specific conditions.

Eg: SELECT \* FROM employees WHERE age > 30;

4. INSERT INTO – Adds new records into a table.

Eg: INSERT INTO employees (name, age, department) VALUES ('Alice', 28, 'HR');

LAB EXERCISES:

Lab 1: Write SQL queries to retrieve specific columns (student\_name and age) from the students table.

Ans. SELECT student\_name, student\_age FROM students WHERE student\_age > 16;

SELECT student\_name, student\_age FROM students ORDER BY student\_age DESC;

Lab 2: Write SQL queries to retrieve all students whose age is greater than 10.

Ans. SELECT \* FROM students WHERE student\_age > 10;

3. SQL Constraints

Theory Questions:

1. What are constraints in SQL? List and explain the different types of constraints.

Ans. Constraints in SQL are rules applied to columns in a database table to enforce data integrity, consistency, and accuracy. Constraints limit the type of data that can be inserted, updated, or deleted in a table, ensuring the validity of the data.

1. NOT NULL Constraint: Ensures that a column cannot have a NULL value. A value must be provided for the column.

2. UNIQUE Constraint: Ensures that all values in a column are unique, meaning no duplicate values are allowed. A column with a unique constraint can still contain NULL values (unless combined with NOT NULL).

3. PRIMARY KEY Constraint: Uniquely identifies each record in a table. It combines both NOT NULL and UNIQUE constraints. A table can have only one primary key, and it may consist of one or multiple columns (composite key).

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

Ans. Keys are one of the most important elements in a relational database to maintain the relationship between the tables and it also helps in uniquely identifying the data from a table. The primary key is a key that helps uniquely identify the tuple of the database. In contrast, the Foreign Key is a key used to determine the relationship between the tables through the primary key of one table that is the primary key of one table acts as a foreign key to another table.

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

Ans. NOT NULL Constraint:

Ensures that a column cannot have NULL values.

Every row must have a valid (non-null) value for this column.

Helps maintain data consistency by preventing missing or undefined values.

UNIQUE Constraint:

Ensures that all values in a column are distinct (no duplicates allowed).

Can be applied to one or multiple columns (when combined, values must be unique together).

Unlike PRIMARY KEY, a table can have multiple UNIQUE columns.

LAB EXERCISES:

Lab 1: 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(255) NOT NULL,

teacher\_subject VARCHAR(255) NOT NULL,

teacher\_email VARCHAR(255) UNIQUE

);

Lab 2: Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table.

Ans.

ALTER TABLE students

ADD COLUMN teacher\_id INT;

ALTER TABLE students

ADD CONSTRAINT fk\_teacher

FOREIGN KEY (teacher\_id) REFERENCES teachers(teacher\_id);

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

Theory Questions:

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

Ans. SQL Data Definition Language (DDL) is a subset of SQL (Structured Query Language) used to define and manage the structure of a database. DDL statements are responsible for creating, altering, and deleting database objects such as tables, indexes, views, and schemas.

2. Explain the CREATE command and its syntax.

Ans. The CREATE command in SQL is a Data Definition Language (DDL) statement used to create new database objects such as tables, indexes, views, and schemas. It is fundamental for establishing the structure of a database and defining how data will be stored and organized.

Syntax: CREATE TABLE table\_name (

column1\_name data\_type [constraints],

column2\_name data\_type [constraints],

...

[table\_constraints]

);

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

Ans. Specifying data types and constraints during table creation in a relational database serves several important purposes. These elements are crucial for ensuring data integrity, optimizing performance, and defining the structure of the database.

LAB EXERCISES:

Lab 1: 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 PRIMARY KEY,

course\_name VARCHAR(255) NOT NULL,

course\_credits INT NOT NULL

);

Lab 2: Use the CREATE command to create a database university\_db.

Ans. CREATE DATABASE university\_db;

5. ALTER Command

Theory Questions:

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

Ans. The ALTER command in SQL is used to make changes to a table, view, or the entire database. We can add, modify, and drop constraints, columns, and indexes using the ALTER command in SQL.

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

Ans. 1. Adding Columns: To add a new column to an existing table, you use the ADD clause.

Syntax: ALTER TABLE table\_name

ADD column\_name data\_type [constraints];

2. Modifying Columns: To modify an existing column, you use the MODIFY (or ALTER COLUMN in some SQL dialects) clause. This can include changing the data type or altering constraints.

Syntax: ALTER TABLE table\_name

MODIFY column\_name new\_data\_type [new\_constraints];

3. Dropping Columns: To remove an existing column from a table, you use the DROP COLUMN clause.

Syntax: ALTER TABLE table\_name

DROP COLUMN column\_name;

LAB EXERCISES:

Lab 1: Modify the courses table by adding a column course\_duration using the ALTER command.

Ans. ALTER TABLE courses

ADD course\_duration INT;

Lab 2: Drop the course\_credits column from the courses table.

Ans. ALTER TABLE courses

DROP COLUMN course\_credits;

6. DROP Command

Theory Questions:

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

Ans. In SQL, the DROP command is used to permanently remove an object from a database, such as a table, database, index, or view. When we DROP a table, both the data and the structure of the object are permanently removed from the database leaving no trace of the object.

Syntax: DROP object object\_name ;

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

Ans. Dropping a table invalidates dependent objects and removes object privileges on the table. If you want to re-create the table, then you must regrant object privileges on the table, re-create the indexes, integrity constraints, and triggers for the table, and respecify its storage parameters.

LAB EXERCISES:

Lab 1: Drop the teachers table from the school\_db database.

Ans.

ALTER TABLE students

DROP FOREIGN KEY fk\_teacher\_id;

DROP TABLE school\_db.teachers;

Lab 2: Drop the students table from the school\_db database and verify that the table has been removed.

Ans. DROP TABLE school\_db.students;

SHOW TABLES IN school\_db;

7. Data Manipulation Language (DML)

Theory Questions:

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

Ans. 1. INSERT – Adds new records to a table.

Syntax: INSERT INTO table\_name (column1, column2, column3)

VALUES (value1, value2, value3);

2. UPDATE – Modifies existing records in a table.

Syntax: UPDATE table\_name

SET column1 = value1, column2 = value2

WHERE condition;

3. DELETE – Removes records from a table.

Syntax: DELETE FROM table\_name

WHERE condition;

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

Ans. The WHERE clause is a critical component of SQL UPDATE and DELETE operations. It specifies the conditions that must be met for the operation to be applied to a particular row or set of rows in a table.

Syntax:UPDATE employees SET salary = salary \* 1.1 WHERE department = 'Sales'; -- Only increases salary for employees in the Sales department

Syntax: DELETE FROM customers WHERE last\_purchase < '2022-01-01'; -- Only deletes customers who haven't made a purchase since 2022

LAB EXERCISES:

Lab 1: Insert three records into the courses table using the INSERT command.

Ans. INSERT INTO courses (course\_id, course\_name, course\_duration) VALUES

(1, "IS", 10),

(2, "DBMS", 15),

(3, "WT", 12);

Lab 2: Update the course duration of a specific course using the UPDATE command.

Ans. UPDATE courses

SET course\_duration = 20

WHERE course\_id = 1;

Lab 3: Delete a course with a specific course\_id from the courses table using the DELETE command.

Ans. DELETE FROM courses

WHERE course\_id = 1;

8. Data Query Language (DQL)

Theory Questions:

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

Ans. The SELECT statement in SQL is used to retrieve data from one or more tables in a database. It is the most commonly used SQL command for querying information.

Syntax: SELECT column1, column2, ...

FROM table\_name

WHERE condition;

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

Ans. WHERE and ORDER BY Clauses in SQL Queries

Both WHERE and ORDER BY clauses are used to refine and organize query results in SQL.

1. WHERE Clause – Filtering Data

The WHERE clause is used to filter records by specifying a condition. Only rows that meet the condition are included in the result.

2. ORDER BY Clause – Sorting Data

The ORDER BY clause sorts query results based on one or more columns in ascending (ASC) or descending (DESC) order.

LAB EXERCISES:

Lab 1: Retrieve all courses from the courses table using the SELECT statement.

Ans. SELECT \* FROM courses;

Lab 2: Sort the courses based on course\_duration in descending order using ORDER BY.

Ans. SELECT \* FROM courses

ORDER BY course\_duration DESC;

Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT.

Ans. SELECT \* FROM courses

ORDER BY course\_duration DESC

LIMIT 2;

9)DCL

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

Ans. GRANT :- It is used to give specific privileges (such as SELECT, INSERT, UPDATE, DELETE) to users or roles on database objects like tables, views, or procedures.

REVOKE :- It is used to remove previously granted privileges from users or roles.

2. How do you manage privileges using these commands?

Ans GRANT privilege ON object TO user [WITH GRANT OPTION];

REVOKE privilege(s) ON object FROM user;

LAB EXERCISES:

Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

1. Create user1 and user2

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

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

2. Grant SELECT Privilege on courses Table to user1

GRANT SELECT ON your\_db.courses TO 'user1'@'localhost';

3. Apply Changes

FLUSH PRIVILEGES;

4. Verify Permissions

SHOW GRANTS FOR 'user1'@'localhost';

Lab 2: Revoke the INSERT permission from user1 and give it to user2.

REVOKE INSERT ON your\_database.courses FROM 'user1'@'localhost';

GRANT INSERT ON your\_database.courses TO 'user2'@'localhost';

SHOW GRANTS FOR 'user1'@'localhost';

SHOW GRANTS FOR 'user2'@'localhost';

10. Transaction Control Language (TCL)

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

Ans. COMMIT:-

Saves all the changes made in the current transaction to the database permanently.

Once a COMMIT is executed, the changes cannot be undone.

ROLLBACK:-

Undoes all changes made in the current transaction if an error occurs or if you change your mind.

2. Explain how transactions are managed in SQL databases.

Ans. A transaction in SQL is a sequence of operations performed as a single logical unit of work. It ensures data integrity and consistency in databases.

LAB EXERCISES:

Lab 1: Insert a few rows into the courses table and use COMMIT to save the changes.

Ans. START TRANSACTION;

INSERT INTO courses (course\_id, course\_name, instructor, duration)

VALUES

(101, 'Database Management', 'Dr. Smith', '12 weeks'),

(102, 'Web Development', 'Prof. Johnson', '10 weeks'),

(103, 'Machine Learning', 'Dr. Brown', '14 weeks');

COMMIT;

Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.

Ans. START TRANSACTION;

-- Insert initial rows

INSERT INTO courses (course\_id, course\_name, instructor, duration)

VALUES

(104, 'Cyber Security', 'Dr. White', '16 weeks'),

(105, 'Cloud Computing', 'Prof. Green', '12 weeks');

COMMIT; -- Save these changes permanently

START TRANSACTION;

-- Insert additional rows

INSERT INTO courses (course\_id, course\_name, instructor, duration)

VALUES

(106, 'Data Science', 'Dr. Black', '14 weeks'),

(107, 'Software Engineering', 'Prof. Blue', '10 weeks');

ROLLBACK;

Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

Ans. START TRANSACTION;

-- Insert some sample courses

INSERT INTO courses (course\_id, course\_name, instructor, duration)

VALUES

(108, 'Artificial Intelligence', 'Dr. Gray', '15 weeks'),

(109, 'Blockchain Technology', 'Prof. Silver', '12 weeks');

COMMIT; -- Save these changes permanently

START TRANSACTION;

-- Create a SAVEPOINT before making updates

SAVEPOINT before\_update;

-- Update course details

UPDATE courses

SET instructor = 'Dr. Gold'

WHERE course\_id = 108;

UPDATE courses

SET duration = '10 weeks'

WHERE course\_id = 109;

-- Rollback to SAVEPOINT (undo only specific updates)

ROLLBACK TO before\_update;

-- Commit remaining changes (if needed)

COMMIT;

11. SQL Joins

1. Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?

Ans. In SQL, a JOIN is used to combine rows from two or more tables based on a related column. It helps retrieve data efficiently by linking information from different tables. The common column between tables is typically a primary key in one table and a foreign key in another.

1. INNER JOIN

Returns only matching records from both tables based on the condition.

If there’s no match, the row is excluded from the result.

2. LEFT JOIN

Returns all records from the left table and matching records from the right table.

If there is no match, NULL values are placed in the columns of the right table.

3. RIGHT JOIN

Returns all records from the right table and matching records from the left table.

If there is no match, NULL values are placed in the columns of the left table.

4. FULL OUTER JOIN

Returns all records when there is a match in either table.

If there’s no match, NULL values appear in the columns of the table without a match.

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

Ans. INNER JOIN: Returns only the rows where there is a match in both tables. If there is no match, the row will not be included in the result.

Eg:

SELECT employees.name, departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id;

LEFT JOIN (or LEFT OUTER JOIN): Returns all the rows from the left table and the matched rows from the right table. If there is no match, NULL values are returned for columns from the right table.

Eg:

SELECT employees.name, departments.department\_name

FROM employees

LEFT JOIN departments ON employees.department\_id = departments.department\_id;

RIGHT JOIN (or RIGHT OUTER JOIN): Similar to the LEFT JOIN, but it returns all the rows from the right table and the matched rows from the left table. If there is no match, NULL values are returned for columns from the left table.

Eg:

SELECT employees.name, departments.department\_name

FROM employees

RIGHT JOIN departments ON employees.department\_id = departments.department\_id;

FULL JOIN (or FULL OUTER JOIN): Returns all the rows from both tables, with matching rows where available. If there is no match, NULL values are returned for columns from the table with no match.

Eg:

SELECT employees.name, departments.department\_name

FROM employees

FULL JOIN departments ON employees.department\_id = departments.department

LAB EXERCISES:

Lab 1: Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

Step 1: Create Tables

CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(50)

);

-- Create the employees table

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(50),

department\_id INT,

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

Step 2: Insert Sample Data

-- Insert data into departments table

INSERT INTO departments (department\_id, department\_name)

VALUES

(1, 'Sales'),

(2, 'Engineering'),

(3, 'HR');

-- Insert data into employees table

INSERT INTO employees (employee\_id, employee\_name, department\_id)

VALUES

(101, 'John Doe', 1),

(102, 'Jane Smith', 2),

(103, 'Emily Davis', 3),

(104, 'Michael Brown', 2);

Step 3: Perform an INNER JOIN

-- Perform INNER JOIN to get employees with their respective departments

SELECT employees.employee\_name, departments.department\_name

FROM employees

INNER JOIN departments ON employees.department\_id = departments.department\_id;

Lab 2: Use a LEFT JOIN to show all departments, even those without employees.

-- Perform LEFT JOIN to get all departments with their employees (including departments with no employees)

SELECT departments.department\_name, employees.employee\_name

FROM departments

LEFT JOIN employees ON departments.department\_id = employees.department\_id;

12. SQL Group By:

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

Ans. 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 total or average of each group. It is commonly used with aggregate functions to perform calculations on groups of data.

Aggregate Functions:

COUNT(): Returns the number of rows in a group.

SUM(): Returns the sum of a numeric column in a group.

AVG(): Returns the average value of a numeric column in a group.

MIN(): Returns the minimum value in a group.

MAX(): Returns the maximum value in a group.

How GROUP BY works:

It groups the result set by one or more columns.

It then applies aggregate functions (like COUNT(), SUM(), etc.) to each group.

2. Explain the difference between GROUP BY and ORDER BY.

LAB EXERCISES:

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

Ans. SELECT department, COUNT(\*) AS number\_of\_employees

FROM employees

GROUP BY department;

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

Ans. SELECT department, AVG(salary) AS average\_salary

FROM employees

GROUP BY department;

13. SQL Stored Procedure

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

Ans. A stored procedure in SQL is a precompiled collection of SQL statements and optional control-flow logic (such as loops and conditions) that can be executed on demand. It is stored in the database and can be invoked using a single call.

Difference from a standard SQL query:

A standard SQL query is a single, one-time query executed against the database to retrieve or manipulate data.

A stored procedure, on the other hand, is a reusable set of SQL queries and logic that can accept parameters, perform complex operations, and return results.

2. Explain the advantages of using stored procedures.

Ans. Performance: Stored procedures are precompiled, meaning the SQL statements are parsed and optimized beforehand, resulting in faster execution.

Code Reusability: Once written, a stored procedure can be executed multiple times, reducing the need for repetitive code.

Security: Stored procedures can help control access to data by limiting direct access to underlying tables. Users can be given permission to execute specific stored procedures rather than allowing direct access to data.

Maintainability: Changes to business logic can be made within the procedure, and once modified, all applications using the procedure benefit from the change without needing modification at each application level.

Error Handling: Stored procedures allow for centralized error handling, making it easier to manage and respond to potential issues.

LAB EXERCISES:

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

DELIMITER $$

CREATE PROCEDURE GetEmployeesByDepartment (IN dept\_name VARCHAR(50))

BEGIN

SELECT \* FROM employees

WHERE department = dept\_name;

END $$

DELIMITER ;

• Lab 2: Write a stored procedure that accepts course\_id as input and returns the course details.

DELIMITER $$

CREATE PROCEDURE GetCourseDetails (IN course\_id INT)

BEGIN

SELECT \* FROM courses

WHERE course\_id = course\_id;

END $$

DELIMITER ;

14. SQL View

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

Ans. A view in SQL is a virtual table that is based on the result of a query. It can include data from one or more tables, and it is essentially a stored query that can be treated like a table in SQL statements.

Difference from a table:

A table physically stores data in the database, whereas a view does not store data. It only stores the SQL query used to retrieve the data.

A table can be modified directly (inserting, updating, deleting rows), whereas a view is read-only in many cases (though this can vary depending on the SQL system).

A view can aggregate or combine data from multiple tables and present it as a single entity, while a table holds data for a single entity.

2. Explain the advantages of using views in SQL databases

Ans. Simplified Querying: Views can simplify complex queries by encapsulating them. Users can select from a view instead of writing long SQL queries repeatedly.

Security: Views can restrict access to certain columns or rows in a table, thus protecting sensitive data.

Data Abstraction: Views provide a way to abstract the complexity of the underlying database schema, making it easier for users to work with data without knowing its structure.

Consistency: A view provides a consistent interface to the data. Even if the underlying database schema changes, the view can be adjusted accordingly without impacting the users accessing it.

Reusability: Views can be reused in other queries, reducing redundancy in SQL code.

LAB EXERCISES:

• Lab 1: Create a view to show all employees along with their department names.\

Ans. CREATE VIEW EmployeeDepartmentView AS

SELECT e.employee\_id, e.name, e.salary, e.department\_id, d.department\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id;

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

Ans. CREATE OR REPLACE VIEW EmployeeDepartmentView AS

SELECT e.employee\_id, e.name, e.salary, e.department\_id, d.department\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

WHERE e.salary >= 50000;

15. SQL Triggers:

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

Ans. A trigger in SQL is a set of SQL statements that automatically execute (or "fire") in response to specific events on a table or view, such as inserts, updates, or deletes. Triggers are used to enforce business rules, validate data, maintain referential integrity, and audit changes in a database.

Types of Triggers:

BEFORE Trigger: Executes before an insert, update, or delete operation is performed on a table. It is used for validation, modification of data before the action, or preventing the operation.

AFTER Trigger: Executes after an insert, update, or delete operation is performed on a table. It is used for actions like logging, auditing, or updating related data in other tables.

INSTEAD OF Trigger: Executes in place of the insert, update, or delete operation. It is used in views to modify data when the view is updated.

When Triggers are Used:

Data validation: Ensuring that data conforms to specific rules before it is inserted or updated.

Audit logging: Tracking changes made to the data, such as recording the previous state of records.

Referential integrity: Maintaining consistency between related tables.

Automated actions: Automatically executing business logic such as recalculating totals or updating other tables when data changes.

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

Ans. INSERT Trigger: Fires when a new record is added to a table. It is typically used to perform actions like logging the new entry, validating data, or updating related tables.

UPDATE Trigger: Fires when an existing record is modified. It is commonly used to maintain historical records, log changes, or update related data to reflect changes.

DELETE Trigger: Fires when a record is deleted from a table. It is often used for actions like logging the deleted record, preventing accidental deletion, or updating related data to maintain consistency.

LAB EXERCISES:

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

DELIMITER $$

CREATE TRIGGER AfterEmployeeInsert

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_log (action, employee\_id, name, action\_time)

VALUES ('INSERT', NEW.employee\_id, NEW.name, NOW());

END $$

DELIMITER ;

• Lab 2: Create a trigger to update the last\_modified timestamp whenever an employee record is updated.

DELIMITER $$

CREATE TRIGGER BeforeEmployeeUpdate

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

SET NEW.last\_modified = NOW();

END $$

DELIMITER ;

16. Introduction to PL/SQL

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

PL/SQL (Procedural Language/SQL) is Oracle's procedural extension to SQL. While SQL is a declarative language used to query and manipulate data, PL/SQL adds procedural constructs like loops, conditionals, and variables, allowing for more complex operations and logic within the database.

PL/SQL enables the creation of stored procedures, functions, triggers, and anonymous blocks, providing a way to handle multiple SQL statements, incorporate control structures, and manage exceptions.

How it extends SQL's capabilities:

Procedural Logic: PL/SQL supports control structures like loops, conditional statements (IF-ELSE), and case handling, which SQL alone does not provide.

Variables and Cursors: It allows the use of variables to hold intermediate results and cursors to iterate over query results, enabling more dynamic and flexible operations.

Error Handling: PL/SQL offers advanced exception handling to catch and manage errors that occur during execution.

Block Structure: PL/SQL supports the use of anonymous blocks that combine SQL statements with procedural logic, making it easier to execute complex operations.

2. List and explain the benefits of using PL/SQL.

Improved Performance: PL/SQL allows multiple SQL statements to be executed in a single block, reducing network traffic and improving performance, especially in database-intensive applications.

Modularity: PL/SQL enables the creation of reusable and modular code through stored procedures and functions. This reduces redundancy and makes code easier to maintain.

Security: PL/SQL can encapsulate sensitive business logic in stored procedures, ensuring that only authorized users can access or modify certain operations.

Error Handling: PL/SQL provides structured exception handling, which allows for more robust error management and ensures that applications can gracefully handle unexpected situations.

Transaction Control: PL/SQL allows for managing transactions, making it easy to commit or roll back changes as needed.

Support for Caching and Bulk Processing: PL/SQL supports bulk operations and caching of results, which can greatly enhance performance when dealing with large datasets.

LAB EXERCISES:

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

DECLARE

total\_employees INT;

BEGIN

SELECT COUNT(\*) INTO total\_employees FROM employees;

DBMS\_OUTPUT.PUT\_LINE('Total number of employees: ' || total\_employees);

END;

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

DECLARE

total\_sales NUMBER(10, 2);

BEGIN

SELECT SUM(order\_amount) INTO total\_sales FROM orders;

DBMS\_OUTPUT.PUT\_LINE('Total sales: ' || total\_sales);

END;

17. PL/SQL Control Structures:

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

Control structures in PL/SQL are used to control the flow of execution within a block of code. They allow conditional execution and looping, making PL/SQL more powerful than standard SQL. The main types of control structures are:

Conditional statements (e.g., IF-THEN, CASE)

Loops (e.g., LOOP, WHILE LOOP, FOR LOOP)

IF-THEN Control Structure:

Used for conditional execution. If the condition is TRUE, the code inside the THEN block executes.

LOOP Control Structure:

Used for executing a block of code repeatedly.

Types of loops:

BASIC LOOP – Runs indefinitely unless explicitly exited.

WHILE LOOP – Runs as long as a condition is TRUE.

FOR LOOP – Iterates a fixed number of times.

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

Decision-Making: Conditional structures (IF-THEN, CASE) allow queries to execute different logic based on data values.

Automation: Loops help automate repetitive tasks, such as iterating through records, performing calculations, or updating multiple rows.

Improved Efficiency: Control structures reduce redundancy by allowing logic to be reused within procedures and functions.

Error Handling: They enable structured error handling with EXCEPTION blocks, making queries more robust.

Data Processing: Loops process large datasets efficiently, avoiding the need for multiple manual queries.

LAB EXERCISES:

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

DECLARE

emp\_id employees.employee\_id%TYPE := 101;

emp\_department employees.department%TYPE;

BEGIN

Retrieve the department of the given employee

SELECT department INTO emp\_department

FROM employees

WHERE employee\_id = emp\_id;

Check department and print a message

IF emp\_department = 'HR' THEN

DBMS\_OUTPUT.PUT\_LINE('Employee works in Human Resources.');

ELSIF emp\_department = 'IT' THEN

DBMS\_OUTPUT.PUT\_LINE('Employee works in the IT department.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Employee works in another department.');

END IF;

END;

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

DECLARE

CURSOR emp\_cursor IS

SELECT name FROM employees;

emp\_name employees.name%TYPE;

BEGIN

-- Open cursor and loop through employee records

FOR emp\_record IN emp\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || emp\_record.name);

END LOOP;

END;

18. SQL Cursors

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

A cursor in PL/SQL is a pointer to a memory location that stores the result set of a query. It allows row-by-row processing of query results.

Types of Cursors:

Implicit Cursor: Automatically created by Oracle for single-row queries (like SELECT INTO, INSERT, UPDATE, and DELETE). It does not need to be explicitly defined or opened.

Explicit Cursor: Defined by the programmer when handling multi-row queries. It must be explicitly declared, opened, fetched, and closed.

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

When a query returns multiple rows: Implicit cursors work only for single-row queries, so explicit cursors are required for handling multiple records.

For better control over data retrieval: Explicit cursors allow fetching rows one by one or in batches.

When performance optimization is needed: Explicit cursors can be optimized using BULK COLLECT for faster data retrieval.

For error handling: Explicit cursors provide more control over error handling using attributes like %FOUND, %NOTFOUND, and %ROWCOUNT.

LAB EXERCISES:

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

DECLARE

CURSOR emp\_cursor IS

SELECT employee\_id, name, department, salary FROM employees;

emp\_id employees.employee\_id%TYPE;

emp\_name employees.name%TYPE;

emp\_department employees.department%TYPE;

emp\_salary employees.salary%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO emp\_id, emp\_name, emp\_department, emp\_salary;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('ID: ' || emp\_id || ', Name: ' || emp\_name || ', Dept: ' || emp\_department || ', Salary: ' || emp\_salary);

END LOOP;

CLOSE emp\_cursor;

END;

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

DECLARE

CURSOR course\_cursor IS

SELECT course\_id, course\_name FROM courses;

v\_course\_id courses.course\_id%TYPE;

v\_course\_name courses.course\_name%TYPE;

BEGIN

OPEN course\_cursor;

LOOP

FETCH course\_cursor INTO v\_course\_id, v\_course\_name;

EXIT WHEN course\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Course ID: ' || v\_course\_id || ', Course Name: ' || v\_course\_name);

END LOOP;

CLOSE course\_cursor;

END;

19. Rollback and Commit Savepoint:

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

A SAVEPOINT is a marker within a database transaction that allows partial rollbacks. It helps restore the database to a specific point without undoing the entire transaction.

COMMIT: Saves all changes made in a transaction permanently. After a COMMIT, all savepoints within the transaction are lost.

ROLLBACK: Reverts changes made in a transaction. When used with a savepoint, it undoes only the changes made after that savepoint, without affecting the earlier changes.

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

Error Handling: If an error occurs, savepoints allow partial rollback instead of rolling back the entire transaction.

Complex Transactions: When executing multiple dependent operations, savepoints help preserve the state of successful steps while retrying failed ones.

Batch Processing: Savepoints allow committing part of a batch while rolling back unsuccessful operations.

Multi-Step Processes: When a transaction has multiple stages, using savepoints ensures data consistency if a failure occurs in later stages.

LAB EXERCISES:

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

BEGIN

Insert a new employee:

INSERT INTO employees (employee\_id, name, department, salary)

VALUES (201, 'John Doe', 'IT', 60000);

Create a savepoint:

SAVEPOINT sp1;

Insert another record

INSERT INTO employees (employee\_id, name, department, salary)

VALUES (202, 'Jane Smith', 'HR', 55000);

Rollback to savepoint sp1 (undoes Jane Smith's record but keeps John Doe's)

ROLLBACK TO sp1;

Commit the transaction (saves John Doe's record permanently)

COMMIT;

END;

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

BEGIN

-- Insert a new department

INSERT INTO departments (dept\_id, dept\_name)

VALUES (101, 'Marketing');

-- Create a savepoint

SAVEPOINT sp1;

-- Insert another department

INSERT INTO departments (dept\_id, dept\_name)

VALUES (102, 'Finance');

-- Commit only the first transaction (Marketing)

COMMIT;

-- Insert another department

INSERT INTO departments (dept\_id, dept\_name)

VALUES (103, 'Operations');

-- Rollback to savepoint sp1 (removes Finance and Operations)

ROLLBACK TO sp1;

-- Commit the transaction (ensures that only Marketing is saved)

COMMIT;

END;