**Module 4 – Introduction to DBMS**

**Theory Questions**

**1.Introduction to SQL:**

**Q-1: - What is SQL, and why is it essential in database management?**

**SQL: - (Structured Query Language) is a standard programming language specifically designed for managing and manipulating relational databases.**

**Why it is essential:**

1. **Data Manipulation: - SQL is used to insert, update, delete, and retrieve data from databases.**
2. **Data Definition: -** **It defines the structure of database tables using commands like CREATE, ALTER, and DROP.**
3. **Data Querying: -** **SQL allows you to query data efficiently using SELECT, WERE, ORDER BY, etc.**
4. **Data Control: -** **SQL manages permissions and access with commands like GRANT and REVOKE.**
5. **Transaction Control: -** **Ensures data integrity with commands like COMMIT, ROLLBACK, and SAVEPOINT.**
6. **Standard Language: -** **SQL is an international standard supported by all major RDBMSs like MySQL, PostgreSQL, Oracle, and SQL Server.**
7. **Scalability & Flexibility: -** **SQL handles large amounts of data and complex queries with ease.**

**Q-2 Difference between DBMS & RDBMS**

|  |  |  |
| --- | --- | --- |
| **S.no** | **DBMS** | **RDBMS** |
| **1** | **In DBMS, data are stored as a file.** | **In RDBMS, data are stored in a tabular form.** |
| **2** | **Relationships between tables/files are maintained programmatically.** | **Relationships can be defined at the time of table creation.** |
| **3** | **DBMS does not support client-server architecture.** | **RDBMS supports client-server architecture.** |
| **4** | |  | | --- | |  |  |  | | --- | | **DBMS does not support distributed architecture.** | | **RDBMS supports distributed architecture.** |
| **5** | **No security data is provided.** | **RDBMS provides multiple levels of security (logging, command, object level).** |
| **6** | **May satisfy fewer than 7 rules of Dr. E. F. Codd.** | **Usually satisfies more than 7–8 rules of Dr. E. F. Codd.** |
| **7** | **Normalization is not available.** | **Normalization is available.** |
| **8** | **Examples: File systems, XML, etc.** | **Examples: MySQL, PostgreSQL, SQL Server, Oracle, etc.** |

**Q-3: - Describe the role of SQL in managing relational databases**

**SQL ROLES: - SQL (Structured Query Language) plays a central role in managing relational databases. It is the standard language used to interact with and manipulate data stored in relational database systems like MySQL, PostgreSQL, SQL Server, Oracle, etc.**

**KEY ROLES: -**

1. **Data Definition (DDL – Data Definition Language):** **SQL is used to define and modify the structure of database objects such as tables, indexes, and views.**

**EX:**

* **CREATE TABLE – create a new table.**
* **ALTER TABLE – modify an existing table.**
* **DROP TABLE – delete a table.**

1. **Data Manipulation (DML – Data Manipulation Language):** **SQL allows inserting, updating, and deleting data within tables.**

**EX:**

* **INSERT INTO – add new records.**
* **UPDATE – modify existing records.**
* **DELETE – remove records.**

1. **Data Retrieval (DQL – Data Query Language):**

**Ex: -**

* **QL enables querying data using the SELECT statement.**
* **It supports filtering (WHERE), sorting (ORDER BY), grouping (GROUP BY), and joining tables.**
* **SELECT \* FROM Students WHERE Age > 18.**

1. **Data Control (DCL – Data Control Language):** **SQL helps manage access and permissions for users.**

**EX:**

* **GRANT – give user privileges.**
* **REVOKE – take away privileges.**

1. **Transaction Control (TCL – Transaction Control Language):** **SQL manages transactions to ensure data integrity.**

**EX:**

* **BEGIN, COMMIT – to start and confirm a transaction.**
* **ROLLBACK– to undo changes if an error occurs.**

**Q-4: - What are the key features of SQL?**

1. **Data Definition Language (DDL): SQL allows you to define the structure of a database.**

**Command:**

* **CREATE – Create tables, views, schemas**
* **ALTER – Modify existing tables**
* **DROP – Delete tables or databases**
* **TRUNCATE – Remove all records quickly**
* **Rename – Changes the name of a table or a column**

1. **Data Manipulation Language (DML):** **Used to manage data in tables (insert, update, delete).**

**Command:**

* **INSERT – Add new data**
* **UPDATE – Modify existing data**
* **DELETE – Remove data**

1. **DQL (Data Query Language) in SQL: DQL is a subset of SQL that is used only to query data from a database. It helps in retrieving and analyzing data without modifying it.**

**Command:**

* **Select- select the row or column**

1. **Data Control Language (DCL):** **Controls access to data in the database.**

**Command:**

* **COMMIT – Save changes permanently**
* **ROLLBACK – Undo changes**

**2. SQL Syntax**

**Q-1 What are the basic components of SQL syntax?**

1. **Statements – Instructions for the database.  
   Examples: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP**
2. **Clauses – Parts of statements that define what to do.  
   Examples: WHERE, FROM, GROUP BY, ORDER BY, HAVING.**
3. **Expressions – Combinations of values and operators.  
   Example: price > 100 AND quantity < 50**
4. **Predicates – Conditions used in statements (usually in WHERE).  
   Example: name = 'John'**
5. **Identifiers – Names of database objects.  
   Examples: table names, column names like students, student\_id.**
6. **Operators – Symbols to perform operations.  
   Examples: =, <, >, LIKE, IN, BETWEEN, AND, OR.**
7. **Functions – Built-in methods to perform operations.  
   Examples: COUNT (), SUM (), AVG (), NOW ()**
8. **Comments – Notes ignored by SQL engine.**

* **Single line: -- comment here**
* **Multi-line: /\* comment block \*/**

**Q-2:** - **Write the general structure of an SQL SELECT statement**

* **SELECT: Specifies the columns to retrieve.**
* **FROM: Specifies the table to retrieve data from.**
* **WHERE *(optional)*: Filters rows based on a condition.**
* **GROUP BY *(optional)*: Groups rows sharing a property so aggregate functions can be applied.**
* **HAVING *(optional)*: Filters groups after grouping (used with GROUP BY).**
* **ORDER BY *(optional)*: Sorts the result by one or more columns.**
* **LIMIT *(optional)*: Limits the number of rows returned.**

**EX: -**

**SELECT Column1, Column2, …**

**FROM Table name**

**WHERE Condition**

**Group By Column**

**HAVING Condition**

**ORDER BY Column ASC/DESC**

**Limit Number.**

**Q-3: - Explain the role of clauses in SQL statements.**

* **Select: - Specifies the columns to be retrieved from a table**
* **From: - indicates the tables from which to retrieve data**
* **Where: - filters rows based on specified conditions**
* **Group By: - Groups Rows that have the same values in specified columns useful with aggregate functions (sum, avg)**
* **Having: filters Groups based on Conditions**
* **Ordered By: - sorts the results set by one or more columns**
* **Inserted into: - Adds new rows into tables**
* **Values: - specifies the values to insert in the new rows.**
* **Updates: - Modifies existing records in the tables**
* **Set: - specifies the columns and values to be updated.**
* **Deleted: - Removes records from a table**
* **Joins: - Combines Rows from two or more tables on a related column**

**3.SQL Constraints: -**

**Q-1: - What are constraints in SQL? List and explain the different types of constraints**

**SQL, constraint keys are specific types of constraints that help maintain the uniqueness, relationship, and integrity of data between tables.**

**Types of constraints: -**

* **Primary Key**
* **Unique Key**
* **Foreign Key**

1. **Primary key:**

* **A Primary key is a column of table which uniquely identifies each**
* **Row in that table**
* **Primary key enforces integrity constraints to the table**
* **Only one primary key is allowed to be used on the table.**
* **The primary key does not accept the nay duplicate and null values.**
* **The primary key value in a table change very rarely so it is chosen with care.**
* **Where the changes can occur in a seldom manner**
* **A primary key of one table can be referenced by foreign key of another table.**

1. **Unique Key:**

* **Unique key constraints also identify an individual table uniquely in a relation or table.**
* **A table can have more than one unique kye primary key,**
* **Unique key constraints can accept only one null value for column.**
* **Unique constraints are also referenced by the foreign key of another table.**

1. **Foreign Key:**

* **When “one table is primary key filed is added to a related “many” tables in order to create the common filed which relates the two tables it is called a foreign key in the “many” table**
* **Ex: salary of employee table to identify the salary of “foreign key column “employee\_id\_Ref ” which refers “Employee” filed in the Employee table**
* **Of “john” is stored in “salary” table but his employee info.**
* **For example, salary hon", his "employee id" is stored with each salary record.**

|  |  |  |
| --- | --- | --- |
|  | Table Employee | |
| Employee | | Employee name |
| 1 | | John |
| 2 | | Alex |
| 3 | | Roy |
| 4 | | Joy |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Table salary | | | |  |  |
| Emp\_ref\_id | | Year | Months | Salary | | |
| 1 | | 2012 | April | 30000 | | |
| 1 | | 2012 | May | 40000 | | |
| 1 | | 2012 | June | 50000 | | |
| 2 | | 2012 | April | 90000 | | |
| 2 | | 2012 | May | 60000 | | |
| 2 | | 2012 | June | 70000 | | |

**Q-2: How do PRIMARY KEY and FOREIGN KEY constraints differ?**

1. **Primary key:**

* **Purpose: uniquely identify each record in a table**
* **Uniqueness: must be unique and not null.**
* **Single Table: Defined within the table it belongs to.**
* **Only Once: A table can have only one primary key.**

**Ex:**

**CREATE TABLE Students (**

**Student\_id INT PRIMARY KEY,**

**Name VARCHAR (50)**

1. **Foreign key:**

* **Purpose: Enforces a link between the data in two tables.**
* **Reference: refers to the primary key in another table**
* **Duplicate Allowed: can have duplicate values**
* **Null Allowed: can be null**
* **Multiple Allowed: A table can have multiple foreign keys**

**Ex:**

**CREATE TABLE Enrollments (**

**enrollment\_id INT PRIMARY KEY,**

**student\_id INT,**

**FOREIGN KEY (student\_id) REFERENCES Students(student\_id)**

**);**

**Q-3: What is the role of NOT NULL and UNIQUE constraints?**

1. **NOT NULL Constraint: Ensure that a column cannot have a Null value.**

* **It is used when a column must always have value-for example user’s email or name should never be left blank.**

**EX:**

**CREATE TABLE users (**

**user\_id INT,**

**username VARCHAR (50) NOT NULL**

**);**

1. **UNIQUE Constraint: Ensure that all values in a column are different and not duplicate**

* **It’s used for fields where each entry should be distinct-like email addresses, usernames, or phone numbers.**

**Ex:**

**CREATE TABLE users (**

**user\_id INT,**

**email VARCHAR (100) UNIQUE**

**);**

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

**Q-1: -** **Define the SQL Data Definition Language (DDL).**

* **Full form: Data Defination Language:**
* **Defination: - DDL is a category of SQL commands used to define, modify, and manage the structure of database objects such as tables, schemas, indexes and constraints.**
* **Features**
* **DDL is focused on the structure of the database**
* **Commands are auto committed**
* **Affects the schema of the database**
* **Commands:**
* **Create: - creates a new table database, index, or view**
* **Alter: - Modifies an existing database object**
* **Drop: - Delete’s objects from the database**
* **Truncate: - Removes all records from a table**
* **Rename: - Changes the name of a database**

**Q-2: - Explain the CREATE command and its syntax.**

**The create command is a data defination language command in sql it is used to create a new database object like:**

* **Tables**
* **Database**
* **Views**
* **Indexes**
* **Schemas**
* **Triggers**
* **Stored procedures….**

**Syntax:**

**CREATE TABLE name (**

**column1 datatype,**

**column2 datatype,**

**column3 datatype**

**Q-3: -** **What is the purpose of specifying data types and constraints during table creation?**

1. **Datatype:**

* **Defines the kind of data each column can hold  
  e.g., INT, VARCHAR, DATE, etc.**
* **Ensure data integrity**
* **optimizes storage and performance**
* **Helps with validation and calculations**

1. **Constraints:**
2. **Maintains data accuracy and integrity**
3. **Enforces rules for the database at the column or table level**

**EX:**

* **NOT NULL: Value must be provided**
* **UNIQUE: No duplicate values**
* **PRIMARY KEY: Uniquely identifies each record**
* **FOREIGN KEY: Maintains relationships between tables**
* **CHECK: Ensures values meet a specific**

**5.Command.**

**Q-1: - What is the use of the ALTER command in SQL?**

1. **Add a column**

**Ex: ALTER TABLE students ADD age INT.**

1. **Drop a Column:**

**Ex: ALTER TABLE students DROP COLUMN age.**

1. **Rename a column:**

**Ex: ALTER TABLE students RENAME COLUMN age TO student age.**

1. **Modify a column’s data type:**

**Ex: ALTER TABLE students MODIFY age VARCHAR (10).**

1. **Drop a constraint:**

**EX: ALTER TABLE students DROP CONSTRAINT pk\_id.**

1. **Rename a table:**

**Ex: ALTER TABLE students RENAME TO student\_info.**

1. **ADD a constraint:**

**Ex: ALTER TABLE students ADD CONSTRAINT pk\_id PRIMARY KEY (id).**

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

1. **Add a Column:**

**ALTER TABLE table\_name**

**ADD column\_name data\_type.**

1. **Modify a Column: -**

**ALTER TABLE table\_name**

**MODIFY column\_name new\_data\_type.**

1. **Drop a Column:**

**ALTER TABLE table\_name**

**DROP COLUMN column\_name**.

**6.DROP Command**

**Q-1:** **What is the function of the DROP command in SQL?**

* **Deletes entire tables**
* **Removes complete databases**
* **Eliminates views, indexes, or other objects**
* **Delets structure and data permanently**
* **Removes all associated constraints and dependencies**

**Syntax:**

1. **Table drop:**

**DROP TABLE table\_name.**

1. **Database drop:**

**DROP DATABASE database\_name.**

**Q-2: -** **What are the implications of dropping a table from a database?**

1. **Permanent Data Loss:**

* **All the data stored in the table is permanently deleted.**
* **Once dropped the table and its data cannot be recovered unless a backup exists.**

1. **Loss of structure:**

* **The table defination is removed.**
* **Any indexes, triggers and rules associated with the table are also deleted.**

1. **Impact on Relationships**

* **If the table is referenced by foreign keys in other tables, dropping it can cause referential integrity issues or errors.**

1. **Broken Dependencies:**

* **Any views, stored procedures, or functions using the dropped table will no longer work and may cause runtime errors.**

**6.Data Manipulation Language (DML)**

**Q-1: - Define the INSERT, UPDATE, and DELETE commands in SQL**

1. **INSERT Command: Add new rows to a table**

**Syntax:**

**INSERT INTO table\_name (column1, column2, ...)**

**VALUES (value1, value2, ...);**

1. **Update: To modify existing data in a table:**

**Syntax:**

**UPDATE table\_name**

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

**WHERE condition.**

1. **Delete: To remove existing rows from a table**

**DELETE FROM table\_name**

**WHERE condition.**

**Q-2: -** **What is the importance of the WHERE clause in UPDATE and DELETE operations?**

* **The WHERE clause is very important in UPDATE and DELETE operations in SQL because it specifies which rows should be updated or deleted. Without it, all rows in the table will be affected, which can lead to unintended data loss or incorrect updates.**

1. **Importance in UPDATE**

* **It ensures only specific records are updated based on a condition.**
* **Prevents updating every row in the table accidentally.**

**Ex:**

**UPDATE students**

**SET age = 18**

**WHERE student\_id = 101.**

1. **importance in DELETE**

* **It ensures only the desired rows are deleted.**
* **Without it, entire tables can be wiped out.**

**Ex:**

**DELETE FROM students**

**WHERE student\_id = 101.**

**7.Data Query Language (DQL)**

**Q-1: - What is the SELECT statement, and how is it used to query data?**

**The SELECT statement is a SQL command used to retrieve data from a table in a database. It allows you to fetch specific columns or all the columns from one or more tables.**

**How to Use the SELECT Statement:**

* **Syntax:**

**SELECT column1, column2,**

**FROM table\_name.**

* **To Select All Columns:**

**SELECT \* FROM table\_name.**

**Example:**

| **student\_id** | **name** | **age** |
| --- | --- | --- |
| **1** | **Anil** | **15** |
|  |  |  |

**Q-2: - Explain the use of the ORDER BY and WHERE clauses in SQL queries1)**

* **WHERE Clause: - The WHERE clause is used to filter records in a query.**
* **It allows you to specify conditions so that only rows that meet those conditions are included in the result.**

**Syntax:**

**SELECT column1, column2**

**FROM table\_name**

**WHERE condition.**

* **ORDER BY Clause: - The ORDER BY clause is used to sort the result set by one or more columns.**
* **Ascending order (ASC)**
* **Descending order (DESC)**

**9. Data Control Language (DCL)**

**Q-1: - What is the purpose of GRANT and REVOKE in SQL?**

* **Grant: - The GRANT command is used to give permissions (privileges) to users on database objects.**

**Ex: -**

**GRANT SELECT, INSERT ON students TO user\_name.**

* **Revoke: - The REVOKE command is used to remove previously granted permissions from users.**

**Ex: -**

**REVOKE UPDATE ON students FROM john.**

**Q-2: - How do you manage privileges using these commands?**

* **GRANT Command: - Used to give privileges to a user.**

**Ex: -**

**GRANT privilege\_type ON object\_name TO user\_name.**

* **REVOKE Command: - Used to take back privileges from a user.**

**Ex: -**

**REVOKE privilege\_type ON object\_name FROM user\_name.**

**10. Transaction Control Language (TCL)**

**Q-1: - What is the purpose of the COMMIT and ROLLBACK commands in SQL?**

**COMMIT and ROLLBACK are transaction control commands used to manage changes made by Data Manipulation Language (DML) operations like INSERT, UPDATE, and DELETE.**

1. **COMMIT Command: - The COMMIT command is used to permanently save all the changes made during the current transaction to the database.**

**Ex:**

**BEGIN.**

**UPDATE accounts SET balance = balance - 1000 WHERE account\_id = 1;**

**UPDATE accounts SET balance = balance + 1000 WHERE account\_id = 2;**

**COMMIT.**

1. **ROLLBACK Command: - The ROLLBACK command is used to undo all changes made in the current transaction before they are committed.**

**Ex: -**

**BEGIN.**

**UPDATE accounts SET balance = balance - 1000 WHERE account\_id = 1;**

**-- Suppose an error occurs here (e.g., account\_id = 2 doesn't exist)**

**ROLLBACK.**

**Q-2:- Explain how transactions are managed in SQL databases.**

**A transaction in SQL is a sequence of one or more SQL operations executed as a single unit of work. It ensures data integrity, consistency, and follows ACID properties:**

* **Atomicity: All operations succeed or none.**
* **Consistency: Data remains valid before and after the transaction.**
* **Isolation: Transactions don’t interfere with each other.**
* **Durability: Once committed, changes are permanent.**

**Ex:**

**Bank transfer:**

**START TRANSACTION.**

**UPDATE accounts SET balance = balance - 500 WHERE acc\_no = 'A’.**

**UPDATE accounts SET balance = balance + 500 WHERE acc\_no = 'B’.**

**COMMIT.**

**11.SQL Joins**

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

**Concept:** **JOIN is used to combine rows from two or more tables based on a related column between them — usually a foreign key and primary key relationship.**

**Types of joints:**

1. **Inner join:**
2. **Left join:**
3. **Right table:**
4. **Full outer join:** 
   1. **Inner join: Returns rows where there is a match on both tables.**

**SELECT s. student name, c. course name**

**FROM students s**

**INNER JOIN courses c ON s. course\_id = c. course\_id.**

* 1. **Left join:** **Returns all rows from the left table, and the matched rows from the right table.  
     If there’s no match, NULL is returned from the right side.**

**SELECT s. student name, c. course name**

**FROM students s**

**LEFT JOIN courses c ON s. course\_id = c. course\_id.**

* 1. **Right join:** **Returns all rows from the right table, and the matched rows from the left table.  
     If there’s no match, NULL is returned from the left side.**

**SELECT s. student name, c. course name**

**FROM students s**

**RIGHT JOIN courses c ON s. course\_id = c. course\_id.**

* 1. **Full outer joint: Returns all rows from both tables, whether or not there is a match.  
     Unmatched fields will contain NULLs.**

**SELECT s. student name, c. course name**

**FROM students s**

**FULL OUTER JOIN courses c ON s. course\_id = c. course\_id.**

**Q-2: How are joins used to combine data from multiple tables**

**JOINs are used to combine data from two or more tables based on a related column between them — usually a primary key in one table and a foreign key in another.**

**Ex:**

**Table – 1**

| **student\_id** | **name** |
| --- | --- |
| **1** | **Aditi** |
| **2** | **Rahul** |
| **3** | **Priya** |

**Table – 2**

| **mark\_id** | **student\_id** | **score** |
| --- | --- | --- |
| **1** | **1** | **85** |
| **2** | **2** | **90** |
| **3** | **4** | **78** |

**Syntax:**

**SELECT students.name, marks. Score**

**FROM students**

**INNER JOIN marks**

**ON students.student\_id = marks.student\_id;**

**Output:**

| **name** | **score** |
| --- | --- |
| **Aditi** | **85** |
| **Rahul** | **90** |

**12.SQL Group By**

**Q-1: What is the GROUP BY clause in SQL? How is it used with aggregate functions?**

**The GROUP BY clause in SQL is used to group rows that have the same values in one or more columns. It is typically used in combination with aggregate functions (such as COUNT (), SUM (), AVG (), MAX (), MIN ()) to perform calculations on each group of data.**

**How is it used with aggregate functions?**

**GROUP BY is used with aggregate functions, SQL groups the rows based on the column(s) specified and then applies the aggregate function to each group separately. This helps in summarizing or analyzing grouped data.**

**Q-2: Explain the difference between GROUP BY and ORDER BY.**

| **Aspect** | **GROUP BY** | **ORDER BY** |
| --- | --- | --- |
| **Purpose** | **Groups rows based on one or more columns** | **Shorts the result set** |
| **Used With** | **Aggregate functions (COUNT (), SUM (), etc.)** | **Any column (with or without aggregate functions)** |
| **Output** | **One row per group** | **All rows, sorted** |
| **Required?** | **When using aggregation and grouping is needed** | **Optional, only used when sorting is needed** |
| **Example** | **GROUP BY department** | **ORDER BY salary DESC** |

**13.SQL Stored Procedure**

**Q-1: What is a stored procedure in SQL, and how does it differ from a standard SQL query?**

**What is a Stored Procedure: A stored procedure in SQL is a precompiled collection of one or more SQL statements that are stored and executed on the database server. It allows you to encapsulate logic such as queries, loops, conditions, and transactions into a single unit that can be executed with a simple call.**

**Stored procedures can:**

* **Accept input parameters,**
* **Return output parameters or result sets,**
* **Include control-of-flow logic (IF, WHILE etc.),**
* **Be reused and maintained easily.**

**How it can differentiate:**

1. **Execution Context:**

* **Stored Procedure: Stored and executed on the database server as a precompiled block of code.**
* **SQL Query: Written and executed manually each time from the client or application.**

1. **Reusability:**

* **Stored Procedure: Can be reused multiple times without rewriting the logic.**
* **SQL Query: Needs to be rewritten or reissued every time it's used.**

1. **Complexity:**

* **Stored Procedure: Supports control flow (like IF, WHILE loops), allowing complex operations.**
* **SQL Query: Typically handles only a single operation (like SELECT, INSERT, etc.).**

1. **Security:**

* **Stored Procedure: Allows better control over access (users can be restricted to execute the procedure without accessing underlying tables directly).**
* **SQL Query: Users need direct access to the tables involved.**

1. **Performance:**

* **Stored Procedure: Executes faster because it is precompiled and cached by the database.**
* **SQL Query: Compiled and executed each time, which can be slower.**

**Q-2: - Explain the advantages of using stored procedures**

1. **Performance Improvement:**

* Stored procedures are compiled and stored in the database, so they execute faster than individual SQL queries sent from applications.

1. **Reusability:**

* Once created, stored procedures can be reused by multiple programs or users without rewriting the SQL logic.

1. **Security:**

* Users can be granted permission to execute the stored procedure without giving direct access to the underlying tables, protecting sensitive data.

1. **Reduced Network Traffic:**

* Only the procedure call is sent from the application to the server, not the entire SQL query, reducing network load.

1. **Easier Maintenance:**

* Any changes to the logic can be made in one place (inside the procedure), without changing all client applications.

1. **Modularity:**

* Complex operations can be broken into smaller procedures, making the code easier to understand and manage.

1. **Data Integrity and Validation:**

* Business logic and validation rules can be enforced at the database level, ensuring consistency across all applications.

1. **Transaction Support:**

* Stored procedures can manage transactions (BEGIN, COMMIT, ROLLBACK) to ensure data integrity in multi-step operations.

**14. SQL View**

**Q-1: What is a view in SQL, and how is it different from a table?**

**What is view SQL: A view in SQL is a virtual table that is based on the result of a SQL query. It does not store data physically, but instead displays data stored in other tables. You can use views to simplify complex queries, enhance security, or present data in a specific format.**

**Ex:**

**CREATE VIEW student\_view AS**

**SELECT name, grade**

**FROM students**

**WHERE grade > 80;**

**How is a View Different from a Table?**

* **Defination**
* **View: Defined by a SQL SELECT statement**
* **Table: Defined by a table schema.**
* **Storage:**
* **View: Does not store data physically**
* **Table:** **Stores data physically in the database**
* **Data source:**
* **View: Retrieves data from one or more tables**
* **Table:** **Is the actual source of data**
* **Modifiable:**
* **View: sometimes read only or updatable**
* **Table:** **Fully modifiable**

**Q-2:** **Explain the advantages of using views in SQL databases’**

* **Keeps data safe – Only selected data is shown to users.**
* **Hides table details – Users don’t need to know table structure.**
* **Easy to manage changes – Changes in table won’t affect users.**
* **Can use again – Views can be used many times.**
* **Gives same result every time – Useful for reports and fixed outputs.**
* **Makes queries simple – Big queries can be saved and reused easily.**

**15. SQL Triggers**

**Q-1: - What is the trigger in SQL? Describe its types and when they are used.**

**What is trigger: A trigger is a set of instructions that automatically runs (fires) when an event (like INSERT, UPDATE, or DELETE) happens on a table.**

**Types of Triggers:**

1. **BEFORE Trigger**

* **Running before the action (insert/update/delete) happens.**
* **it Used to check data or modify it before saving.**

1. **AFTER Trigger**

* **Running after the action is completed.**
* **it Used to log changes, update other tables, etc.**

1. **INSTEAD OF Trigger**

* **Used with views (not tables)**
* **it Used to replace the action (e.g., perform custom action instead of default insert/update)**

**When trigger used:**

1. **Audit Logging**: To keep a record of data changes (who changed what and when).  
   Example: Save old data in a log table before it's updated.
2. **Data Validation**: To check or block wrong data before it's saved.  
    Example: Stop inserting a record if a value is too high.
3. **automatic Updates**: Update related tables automatically.  
    Example: Reduce product stock after a sale.
4. **Enforcing Business Rules**: To apply rules that aren’t possible with just constraints.  
    Example: Only allow updates during working hours.
5. **Prevent Invalid Actions**: Stop actions that may harm the database.  
    Example: Prevent deleting a customer if they have pending orders.

**Q-2: Explain the difference between INSERT, UPDATE, and DELETE triggers.**

**1. INSERT Trigger: A new row is inserted into a table.**

* **It is used to log new entries, enforce business rules, or initialize values.**
* **Example**: Automatically log who added the record and when.

**2. UPDATE Trigger:** **An existing row is modified (i.e., column values are changed).**

* **It is used to track changes, validate new data, or restrict certain updates.**
* **Ex Keep a record of old and new salary when employee salary is updated.**

**3. DELETE Trigger: A row is deleted from the table.**

* **It is used to prevent accidental deletion; log deleted data or archive it.**
* **Example: Save deleted student data to a backup table before removal.**

**16. Introduction to PL/SQL**

**Q-1:** **What is PL/SQL, and how does it extend SQL's capabilities?**

**PL/SQL (Procedural Language/Structured Query Language) is Oracle's procedural extension to SQL. It combines the power of SQL with the features of a procedural programming language like loops, conditions, variables, and functions.**

**How PL/SQL Extends SQL's Capabilities:**

|  |  |
| --- | --- |
| **SQL** | **PL/SQ** |
| **SQL is a declarative language – you specify *what* to do (e.g., SELECT, INSERT).** | **PL/SQL is procedural – you can write *how* to do it (step-by-step logic).** |
| **Cannot handle complex logic, loops, or decisions.** | **Allows IF-ELSE, LOOPs, CASE, etc. for complex operations.** |
| **Executes single queries/statements.** | |  | | --- | |  |  |  | | --- | | **Supports blocks of code with multiple statements.** | |
| **No concept of variables, conditions, or functions.** | **Supports variables, constants, procedures, and functions** |
| **Less control over error handling.** | |  | | --- | |  |  |  | | --- | | **Built-in exception handling mechanisms.** | |

**Q-2: - List and explain the benefits of using PL/SQL**

1. **Block Structure**

* PL/SQL code is written in **blocks** (with DECLARE, BEGIN, EXCEPTION, END).
* This structure makes code **organized**, **modular**, and easier to **read and maintain**.

1. **Procedural Capabilities**

* Unlike SQL, PL/SQL supports **loops**, **conditions (IF, CASE)**, and **functions/procedures**.
* You can write **complex logic** in a single program unit.

1. **Better Performance**

* PL/SQL reduces the **number of database calls** by grouping SQL statements.
* This improves **execution speed** and reduces **network traffic**.

1. **Error Handling**

* It includes **EXCEPTION handling** to catch and handle runtime errors.
* programs make them more **robust and reliable**.

1. **Reusability**

* You can create **procedures, functions, packages**, and **triggers**.
* These can be reused in multiple programs or applications, saving time and effort.

1. **Tight Integration with SQL**

* PL/SQL is designed to **work closely with SQL**.
* You can use **SQL statements** directly inside PL/SQL code.

1. **Security**

* You can control **user access** to data using PL/SQL procedures and packages.
* Sensitive logic can be hidden from unauthorized users.

1. **Portability**

* PL/SQL code can run on any **Oracle database system** without change.
* Makes it easier to move applications across environments.

1. **Support for Triggers**

* PL/SQL allows creating **triggers** to automatically perform actions on table changes (INSERT, UPDATE, DELETE).
* Useful for **auditing** and **data integrity**.

1. **Maintainability**

* Because logic is centralized in **procedures/packages**, updates require changing code in one place.
* Simplifies **debugging and maintenance**.

**17. PL/SQL Control Structures**

**Q-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 in a program. They allow decisions to be made and repeated execution of statements**

**There are three main types of control structures:**

* **Conditional control – using IF statements**
* **Iterative control – using LOOP, WHILE, and FOR loops**
* **Sequential control – using GOTO**

1. **IF-THEN Control Structure:** **The IF-THEN structure is used to execute a block of code only if a certain condition is true.**

**Syntax**

**IF condition THEN**

**-- Statements to execute if condition is true**

**END IF;**

**Example:**

**DECLARE**

**marks NUMBER:= 75;**

**BEGIN**

**IF marks > 60 THEN**

**DBMS\_OUTPUT.PUT\_LINE('Passed').**

**END IF;**

**END;**

1. **LOOP Control Structure (Iterative Control): The LOOP structure is used to repeat a block of code indefinitely or until a specific condition is met using an EXIT statement.**

**Syntax**

**LOOP**

**-- Statements**

**EXIT WHEN condition;**

**END LOOP;**

**Example**

**DECLARE**

**i NUMBER := 1;**

**BEGIN**

**LOOP**

**DBMS\_OUTPUT.PUT\_LINE('Value of i: ' || i);**

**i := i + 1;**

**EXIT WHEN i > 5;**

**END LOOP;**

**END;**

**Q-2: How do control structures in PL/SQL help in writing complex queries?**

**Control structures in PL/SQL help in writing complex queries by allowing decision-making, repetition, and conditional execution of SQL and PL/SQL statements. They make your code dynamic, flexible, and more powerful than plain SQL.**

**1. Conditional Logic (IF-THEN-ELSE): You can perform different actions based on conditions.**

**Example: Apply different discounts based on customer type.**

**IF customer\_type = 'Premium' THEN**

**discount := 0.2;**

**ELSE**

**discount := 0.1;**

**END IF;**

**2. Loops (LOOP, WHILE, FOR): Used to repeat actions, such as processing multiple rows or executing a block repeatedly.**

**Example: Give bonuses to employees one by one.**

**FOR I IN 1...10 LOOP**

**give\_bonus(emp\_id\_list(i));**

**END LOOP;**

**3. CASE Statements: Simplifies multiple conditions into a cleaner structure (like switch-case in other languages).**

**Example : Set status based on score range.**

**CASE**

**WHEN score >= 90 THEN grade := 'A';**

**WHEN score >= 80 THEN grade := 'B';**

**ELSE grade := 'C';**

**END CASE;**

**4. Exception Handling :- Manages runtime errors and ensures the program doesn't crash.**

**Example : Handle a division-by-zero error gracefully.**

**BEGIN**

**result := value1 / value2;**

**EXCEPTION**

**WHEN ZERO\_DIVIDE THEN**

**result := 0;**

**END;**

**18. SQL Cursors**

**Q-1: What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors**

**A cursor in PL/SQL is a pointer that retrieves rows returned by a SQL query one at a time. It enables row-by-row processing of query results, especially useful when dealing with multiple rows.**

**Diffrance**

| **Feature** | **Implicit Cursor** | **Explicit Cursor** |
| --- | --- | --- |
| **Declaration** | **Created automatically by PL/SQL** | **Must be declared manually using CURSOR** |
| **Used For** | **Single-row queries (e.g. SELECT INTO) or DML** | **Multi-row SELECT queries** |
| **Control** | **No control over opening, fetching, or closing** | **Full control using OPEN, FETCH, CLOSE** |
| **Attributes** | **Uses SQL%FOUND, SQL%ROWCOUNT, etc.** | **Uses <cursor name>%FOUND, %ROWCOUNT, etc.** |
| **Example Use** | **SELECT INTO, UPDATE, DELETE, INSERT** | **Custom loop processing with FETCH INTO** |

**Q-2:** **When would you use an explicit cursor over an implicit on**

1. **When You Need to Process Multiple Rows One by One**

* **Implicit cursors can only fetch one row (e.g., SELECT INTO).**
* **If your query returns more than one row, using SELECT INTO will cause a TOO\_MANY\_ROWS error**

1. **When You Need More Control Over the Query Execution**

* **With an explicit cursor, you control:**
* **When to open the cursor**
* **When and how to fetch each row**
* **When to exit the loop**
* **When to close the cursor**
* **This control is not available with implicit cursors.**

1. **When You Want to Reuse a Query with Parameters**

* **Explicit cursors can be parameterized. You can pass values into the cursor, making it reusable for different input values.**

1. **When You Need to Use Cursor Attributes During Row Processing**

* **Explicit cursors give better access to attributes like:**
* **%ROWCOUNT: Number of rows fetched so far**
* **%FOUND, %NOTFOUND: Useful after each fetch**

**19. Rollback and Commit Save point**

**Q-1: Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with saving points?**

**Concept of SAVEPOINT in Transaction Management:** **A SAVEPOINT is a marker or checkpoint within a transaction that allows you to partially roll back the transaction to a specific point without affecting the entire transaction.**

**It is used to divide a long transaction into smaller parts so that you can undo a portion of it if needed, rather than rolling back the entire transaction.**

**Syntax:** **SAVEPOINT savepoint\_name;**

* **Interaction with ROLLBACK and COMMIT with SAVEPOINT**

1. **ROLLBACK TO SAVEPOINT:** **If an error occurs after a SAVEPOINT, you can undo all operations after that save point using:**

**ROLLBACK TO savepoint\_name;**

1. **COMMIT: When you issue a COMMIT, all changes made in the transaction, including those before and after saving points, are made permanent.**

**Once committed, you cannot roll back to any save point — they are automatically released**

**Ex:**

**BEGIN;**

**UPDATE savings account**

**SET balance = balance - 1000**

**WHERE account no = 'SAV123';**

**UPDATE credit\_card\_account**

**SET balance = balance + 1000**

**WHERE card no = 'CRD456';**

**COMMIT;**

**Q-2: When is it useful to use save points in a database transaction**

**Save points are useful in a database transaction when you want partial rollback control within a larger transaction. They allow you to mark specific points in a transaction so you can roll back to that point without affecting prior operations.**

**When is it useful to use SAVEPOINT in a transaction?**

1. **Partial Rollback**

* **If an error occurs during a part of the transaction, you can roll back only that part using a save point.**
* **It helps avoid losing the successful earlier operations.**

1. **Complex Transactions**

* **When your transaction includes multiple steps or operations, and only one step might fail.**
* **Save points give you more control.**

1. **Error Handling**

* **You can test risky SQL statements, and if they fail, you can roll back to a previous stable point**

1. **Nested Transactions (Simulated)**

* **In systems that don’t support true nested transactions, saving points can simulate them.**