**Module 4 : Introduction to DBMS**

**Introduction to SQL**

**Theory Questions:**

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

**SQL (Structured Query Language)** is a standard language used to interact with and manage relational databases.  
It helps in performing operations like:

* Creating databases and tables
* Inserting, updating, deleting, and retrieving data
* Managing database permissions

**Why is SQL essential?**

* It makes working with large amounts of data simple and efficient.
* It allows users to retrieve exactly the data they want using queries.
* It provides a structured way to manage and manipulate data in relational databases.

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **DBMS (Database Management System)** | **RDBMS (Relational Database Management System)** |
| **Data Structure** | Data is stored as files (hierarchical or network structure) | Data is stored in tables (rows and columns) |
| **Relationships** | Limited or no relationship between data | Tables are related using primary keys and foreign keys |
| **Data Integrity** | Less enforcement of data integrity | Strong data integrity with constraints (Primary Key, Foreign Key) |
| **Examples** | Microsoft Access, file system | MySQL, Oracle, SQL Server, PostgreSQL |
| **Scalability** | Less scalable for large data | Highly scalable for large and complex data |

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

SQL helps in managing relational databases by allowing users to:

1. **Create and modify database structures** – Create tables, define columns, constraints, etc.
2. **Insert, Update, and Delete Data** – Add new records, update existing records, or delete unwanted data.
3. **Query Data** – Retrieve useful information using SELECT statements with filters, sorting, and grouping.
4. **Control Access** – Manage user permissions to secure the database (GRANT, REVOKE).
5. **Maintain Data Integrity** – Enforce rules (like primary keys, foreign keys) that ensure valid and consistent data.

**Example SQL commands:**

* CREATE TABLE, INSERT INTO, SELECT, UPDATE, DELETE, DROP TABLE

**4. What are the key features of SQL?**

1. **Data Definition Language (DDL)**:  
   Commands to define database structure (e.g., CREATE, ALTER, DROP).
2. **Data Manipulation Language (DML)**:  
   Commands to manipulate data (e.g., INSERT, UPDATE, DELETE, SELECT).
3. **Data Control Language (DCL)**:  
   Commands for access control (e.g., GRANT, REVOKE).
4. **Transaction Control Language (TCL)**:  
   Commands to manage transactions (e.g., COMMIT, ROLLBACK, SAVEPOINT).
5. **Easy Data Retrieval**:  
   Powerful SELECT queries with filtering, joining multiple tables, ordering, grouping.
6. **Data Integrity & Constraints**:  
   Enforces rules like primary key, foreign key, UNIQUE, NOT NULL, CHECK constraints.
7. **Standardized Language**:  
   SQL is widely used across many database systems like MySQL, Oracle, MS SQL Server, PostgreSQL.

**2. SQL Syntax Theory Questions:**

**1. What are the basic components of SQL syntax?**

The basic components of SQL syntax are:

1. **Keywords**  
   Words like SELECT, FROM, WHERE, INSERT INTO, UPDATE, DELETE – These are predefined and always written in uppercase (by convention).
2. **Identifiers**  
   Names of database objects like table names, column names, database name, etc.  
   Example: students, id, name, age
3. **Operators**  
   Symbols used for comparisons or calculations:  
   Example: =, >, <, >=, <=, <>, AND, OR, LIKE
4. **Expressions**  
   Combination of identifiers, constants, operators used to produce a value.  
   Example: age > 18, name LIKE 'A%'
5. **Clauses**  
   Specific parts of an SQL statement (e.g., WHERE, ORDER BY, GROUP BY).
6. **Semicolon (;)**  
   Used to end an SQL statement (optional in some DBMS, but good practice).

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

SELECT column1, column2, ...

FROM table\_name

WHERE condition

ORDER BY column\_name;

**Example:**

SELECT name, age

FROM students

WHERE age >= 18

ORDER BY name ASC;

**Explanation:**

* SELECT: Lists the columns you want to retrieve.
* FROM: Specifies the table.
* WHERE: Filters rows based on condition.
* ORDER BY: Sorts the result.

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

Clauses are the building blocks of an SQL statement that define what operation to perform and how.

Common SQL Clauses:

1. **SELECT**  
   Specifies columns to retrieve from the table.
2. **FROM**  
   Specifies the table name to retrieve data from.
3. **WHERE**  
   Applies conditions to filter rows (like a filter).
4. **ORDER BY**  
   Sorts the result set in ascending (ASC) or descending (DESC) order.
5. **GROUP BY**  
   Groups rows based on a column (useful with aggregate functions like COUNT, SUM).
6. **HAVING**  
   Applies conditions on groups (similar to WHERE but for grouped results).

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

**Clauses** are the building blocks of an SQL statement that define what operation to perform and how.

**Common SQL Clauses:**

1. **SELECT**  
   Specifies columns to retrieve from the table.
2. **FROM**Specifies the table name to retrieve data from.
3. **WHERE**Applies conditions to filter rows (like a filter).
4. **ORDER BY**Sorts the result set in ascending (ASC) or descending (DESC) order.
5. **GROUP BY**Groups rows based on a column (useful with aggregate functions like COUNT, SUM).
6. **HAVING**Applies conditions on groups (similar to WHERE but for grouped results).

**Example Of Multiple Clauses:**

SELECT department, COUNT(\*)

FROM employees

WHERE salary > 50000

GROUP BY department

HAVING COUNT(\*) > 5

ORDER BY department;

**3. SQL Constraints Theory Questions:**

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

**Constraints** are rules applied to table columns to enforce data integrity and accuracy.

They ensure that the data entered in the table follows certain rules and prevents incorrect data entry.

**Types of Constraints in SQL**:

|  |  |
| --- | --- |
| **Constraint** | **Meaning & Purpose** |
| **PRIMARY KEY** | Uniquely identifies each record in a table. It does not allow NULL and duplicate values. |
| **FOREIGN KEY** | Ensures referential integrity by linking a column in one table to the PRIMARY KEY of another table. |
| **NOT NULL** | Ensures that a column cannot have NULL values (must contain data). |
| **UNIQUE** | Ensures that all values in a column are unique (no duplicate values allowed). |
| **CHECK** | Ensures that the value in a column meets a specified condition (e.g., age > 18). |
| **DEFAULT** | Assigns a default value to a column when no value is specified during insertion. |

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **PRIMARY KEY** | **FOREIGN KEY** |
| **Purpose** | Uniquely identifies each row in its own table. | Establishes a relationship between two tables by referencing the PRIMARY KEY of another table. |
| **Uniqueness** | Always unique and NOT NULL. | Can have duplicates and allows NULL (unless specified otherwise). |
| **Example** | student\_id in students table. | student\_id in grades table refers to student\_id in students table. |
| **Role** | Ensures unique records. | Ensures data consistency across tables (referential integrity). |

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

|  |  |
| --- | --- |
| **Constraint** | **Role** |
| **NOT NULL** | Ensures a column always contains a value and cannot be left empty (NULL is not allowed). Example: Every student must have a name entered. |
| **UNIQUE** | Ensures all values in a column are unique across the table, preventing duplicate data. Example: Email column where no two students can have the same email address. |

**Example:**

CREATE TABLE students (

student\_id INT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE

);

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

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

**DDL (Data Definition Language**) is a subset of SQL commands used to define and manage database structures like databases, tables, and indexes.

**Main Purpose of DDL**:  
To create, modify, and delete database objects.

**Common DDL Commands**:

|  |  |
| --- | --- |
| **Command** | **Purpose** |
| **CREATE** | Creates databases, tables, indexes, views, etc. |
| **ALTER** | Modifies the structure of an existing database object (e.g., adding a column). |
| **DROP** | Deletes tables, databases, views, etc. |
| **TRUNCATE** | Removes all data from a table but keeps its structure. |

**2. Explain the CREATE command and its syntax.**

**CREATE Command**:  
Used to create a new database or table.

👉 **Syntax to create a table**:

CREATE TABLE table\_name (

column1 datatype constraint,

column2 datatype constraint,

column3 datatype constraint

);

**Example**:

CREATE TABLE students (

student\_id INT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

age INT,

email VARCHAR(100) UNIQUE

);

**Explanation**:

* students is the table name.
* student\_id is of type INT and is the PRIMARY KEY.
* name is a string (up to 50 characters) and cannot be NULL.
* email must be unique.

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

|  |  |
| --- | --- |
| **Purpose** | **Explanation** |
| **Data Types** | Define what kind of data the column will store (e.g., INT, VARCHAR, DATE, etc.).👉 Example: age INT ensures only integer values are allowed. |
| **Constraints** | Enforce rules on the data to ensure accuracy and consistency.👉 Examples:– PRIMARY KEY ensures unique identification.– NOT NULL ensures no empty values.– UNIQUE avoids duplicate data.– FOREIGN KEY maintains relationships between tables. |

**5. ALTER Command**

**Theory Questions:**

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

The ALTER command is used to modify the structure of an existing table in a database.

**It helps when you want to:**

* Add new columns
* Change data types of columns
* Modify constraints
* Drop (delete) columns

**Purpose**:  
It allows you to change the table structure without deleting the table or its data.

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

**a) Add a Column**

**Syntax**:

ALTER TABLE table\_name

ADD column\_name datatype constraint;

**Example**:

ALTER TABLE students

ADD address VARCHAR(100);

This adds a new column address to the students table.

**b) Modify a Column**

**Syntax** (depends on DBMS):

ALTER TABLE table\_name

MODIFY column\_name new\_datatype new\_constraint;

**Example**:

ALTER TABLE students

MODIFY age SMALLINT;

This changes the age column to type SMALLINT.

**c) Drop a Column**

**Syntax**:

ALTER TABLE table\_name

DROP COLUMN column\_name;

**Example**:

ALTER TABLE students

DROP COLUMN address;

**6. DROP Command**

**Theory Questions:**

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

The DROP command is used to permanently delete a database object, such as a table, database, view, or index.

It removes the entire table structure along with all its data from the database.

**Syntax to drop a table:**

DROP TABLE table\_name**;**

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

|  |  |
| --- | --- |
| **Implication** | **Explanation** |
| **Permanent Data Loss** | All the data inside the table is permanently deleted and cannot be recovered unless there is a backup. |
| **Structure is Removed** | The table structure (columns, constraints, indexes) is completely removed. |
| **Affects Dependent Objects** | If other tables or views depend on this table (e.g., via foreign key), they may stop working or show errors. |
| **No Undo Option** | Once executed, the DROP command cannot be rolled back in many databases unless inside a transaction block (and supported). |

**Example cautionary usage:**

-- Dangerous if done by mistake!

DROP TABLE employees;

--Better to confirm first or use IF EXISTS to prevent errors:

DROP TABLE IF EXISTS employees;

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

**Theory Questions:**

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

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

**INSERT Command**

* Used to **add new records (rows)** into a table.

**Syntax**:

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

VALUES (value1, value2, value3);

**Example**:

INSERT INTO students (student\_id, name, age, email)

VALUES (1, 'Ravi', 20, 'ravi@example.com');

**UPDATE Command**

* Used to modify existing records in a table.

**Syntax**:

UPDATE table\_name

SET column1 = value1, column2 = value2

WHERE condition;

**Example**:

UPDATE students

SET age = 21

WHERE student\_id = 1;

**DELETE Command**

* Used to **remove records (rows)** from a table.

**Syntax**:

DELETE FROM table\_name

WHERE condition;

**Example**:

DELETE FROM students

WHERE student\_id = 1;

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

|  |  |
| --- | --- |
| **Operation** | **Importance of WHERE Clause** |
| **UPDATE** | Without a WHERE clause, all rows will be updated, which is usually not desired. It helps to update only specific rows based on a condition. |
| **DELETE** | Without a WHERE clause, all rows in the table will be deleted, effectively clearing the entire table. The WHERE clause ensures that only selected rows are deleted. |

**Without WHERE (Dangerous):**

UPDATE students SET age = 22;

-- All student ages will be updated to 22 (probably a mistake).

DELETE FROM students;

-- All student records will be deleted (dangerous if unintended).

**With WHERE (Safe):**

UPDATE students SET age = 22 WHERE student\_id = 1;

-- Only updates the student with ID 1.

DELETE FROM students WHERE student\_id = 1;

-- Only deletes the student with ID 1.

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

**Theory Questions:**

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

The SELECT statement is used to retrieve (query) data from one or more tables in a database.

It helps us get exactly the data we want by selecting specific columns, applying conditions, sorting, and grouping.

**General Syntax**:

SELECT column1, column2, ...

FROM table\_name

WHERE condition

ORDER BY column1 ASC|DESC;

**Example**:

SELECT name, age

FROM students

WHERE age >= 18

ORDER BY name ASC;

🔸 **Explanation**:

* Retrieves name and age columns from the students table.
* Filters rows where age is greater than or equal to 18.
* Sorts the result by name in ascending order.

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

|  |  |
| --- | --- |
| **Clause** | **Purpose** |
| **WHERE** | Used to filter records based on specific conditions. It controls which rows are included in the result. |
| **ORDER BY** | Used to sort the result set in ascending (ASC) or descending (DESC) order based on one or more columns. |

Examples:

**Using WHERE:**

SELECT \* FROM students

WHERE age > 20;

* Retrieves only the students whose age is greater than 20.

**Using ORDER BY:**

SELECT name, age FROM students

ORDER BY age DESC;

* Retrieves students sorted by age in descending order (oldest first).

**Combining WHERE and ORDER BY:**

SELECT name, age FROM students

WHERE age > 18

ORDER BY name ASC;

* Retrieves students aged over 18, sorted by name in alphabetical order.

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

**Theory Questions:**

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

DCL (Data Control Language) is used to control access and permissions on database objects like tables.

|  |  |
| --- | --- |
| **Command** | **Purpose** |
| **GRANT** | Gives specific privileges (permissions) to users or roles to access or manipulate database objects. |
| **REVOKE** | Removes previously granted privileges from users or roles, restricting their access. |

**Example of GRANT**:

GRANT SELECT, INSERT ON students TO 'user1';

* Allows user1 to read (SELECT) and insert (INSERT) data in the students table.

**Example of REVOKE**:

REVOKE INSERT ON students FROM 'user1';

* Removes the ability of user1 to insert data into the students table.

**2. How do you manage privileges using these commands?**

**Step 1: Grant Privileges**

Use GRANT to allow users to perform specific actions.

Example:

GRANT SELECT, UPDATE ON students TO 'user1';

* user1 can now select data and update records in the students table.

**Step 2: Revoke Privileges**

If you want to prevent a user from performing actions, use REVOKE.

Example:

REVOKE UPDATE ON students FROM 'user1';

* Now, user1 can no longer update records in the students table, but can still select data (if that privilege was not revoked).

**Important Notes**:

* Privileges can be granted for: SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES, etc.
* Example of giving all privileges:

GRANT ALL PRIVILEGES ON database\_name.\* TO 'user1';

* Always manage privileges carefully to maintain security and data integrity.

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

**Theory Questions:**

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

|  |  |
| --- | --- |
| **Command** | **Purpose** |
| **COMMIT** | Saves all the changes made by the current transaction permanently into the database. Once committed, the changes cannot be undone. |
| **ROLLBACK** | Reverts all the changes made by the current transaction, bringing the database back to its previous state before the transaction started. |

👉 **Example of COMMIT**:

INSERT INTO students (student\_id, name, age)

VALUES (1, 'Amit', 21);

COMMIT;

* The new student record is saved permanently in the database.

👉 **Example of ROLLBACK**:

INSERT INTO students (student\_id, name, age)

VALUES (2, 'Rita', 22);

ROLLBACK;

* The new student record is NOT saved. The database remains unchanged.

**2. Explain how transactions are managed in SQL databases.**

A transaction is a sequence of one or more SQL operations (INSERT, UPDATE, DELETE, etc.) treated as a single unit of work.

**Properties of a Transaction (ACID Properties):**

1. **Atomicity** – All steps of the transaction are treated as one unit.  
   ➔ Either all succeed or all fail (nothing partial).
2. **Consistency** – The database must remain in a valid state before and after the transaction.
3. **Isolation** – Transactions are executed independently without interfering with each other.
4. **Durability** – Once committed, changes remain permanently even if there is a system failure.

**How It Works:**

1. Start a transaction (automatically or explicitly).
2. Perform multiple operations (INSERT, UPDATE, DELETE).
3. Use COMMIT to permanently save changes.
4. Or use ROLLBACK to cancel all changes made during the transaction if something goes wrong.

**Example Transaction Workflow:**

BEGIN TRANSACTION;

INSERT INTO students (student\_id, name, age) VALUES (3, 'Sita', 23);

UPDATE students SET age = 24 WHERE student\_id = 3;

-- If everything is correct:

COMMIT;

-- If something goes wrong:

ROLLBACK;

**Why Important?**  
This ensures data integrity and prevents incomplete or corrupted data from being saved.

**11. SQL Joins**

**Theory Questions:**

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

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

👉 **Types of Joins**:

|  |  |  |
| --- | --- | --- |
| **Join Type** | **Description** | **Example Visual (Simplified)** |
| **INNER JOIN** | Returns only the matching rows from both tables. | A ∩ B (Intersection) |
| **LEFT JOIN (LEFT OUTER JOIN)** | Returns all rows from the left table, and matching rows from the right table. If no match, NULL is shown for right table columns. | All of A + matched B |
| **RIGHT JOIN (RIGHT OUTER JOIN)** | Returns all rows from the right table, and matching rows from the left table. If no match, NULL is shown for left table columns. | All of B + matched A |
| **FULL OUTER JOIN** | Returns all rows when there is a match in one of the tables. Non-matched rows from both sides appear with NULL in the opposite side. | A ∪ B |

**Examples**:

**INNER JOIN Example:**

SELECT students.name, grades.marks

FROM students

INNER JOIN grades ON students.student\_id = grades.student\_id;

* Only returns students who have grades.

**LEFT JOIN Example:**

SELECT students.name, grades.marks

FROM students

LEFT JOIN grades ON students.student\_id = grades.student\_id;

* Returns all students, even if they don’t have grades. Marks will be NULL if no matching grade.

**RIGHT JOIN Example:**

SELECT students.name, grades.marks

FROM students

RIGHT JOIN grades ON students.student\_id = grades.student\_id;

* Returns all grades, even if no matching student exists (rare case).

**FULL OUTER JOIN Example:**

SELECT students.name, grades.marks

FROM students

FULL OUTER JOIN grades ON students.student\_id = grades.student\_id;

* Returns all students and all grades, filling in NULL where no match exists.

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

**Joins combine data by connecting rows from multiple tables based on related columns** (usually primary key and foreign key).

Example Scenario:

* Table 1: students  
  | student\_id | name |  
  |------------|--------|  
  | 1 | Amit |  
  | 2 | Sita |
* Table 2: grades  
  | grade\_id | student\_id | marks |  
  |----------|------------|-------|  
  | 1 | 1 | 85 |  
  | 2 | 3 | 90 |

**INNER JOIN Result**:

|  |  |
| --- | --- |
| **name** | **marks** |
| Amit | 85 |

Only rows where students.student\_id = grades.student\_id are included.

**LEFT JOIN Result**:

|  |  |
| --- | --- |
| **name** | **marks** |
| Amit | 85 |
| Sita | NULL |

All students are listed, and grades appear where available.

**Key Point**:  
Joins help combine meaningful information from multiple tables in one result set, which makes data more useful and complete.

**12. SQL Group By**

**Theory Questions:**

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

The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows.

It is mainly used together with **aggregate functions** like:

* COUNT() – Counts the number of rows
* SUM() – Calculates the total sum
* AVG() – Calculates the average
* MAX() – Finds the maximum value
* MIN() – Finds the minimum value

**General Syntax**:

SELECT column\_name, AGGREGATE\_FUNCTION(column\_name)

FROM table\_name

GROUP BY column\_name;

**Example**:

Suppose we have a table grades:

|  |  |  |
| --- | --- | --- |
| **student\_id** | **subject** | **marks** |
| 1 | Math | 80 |
| 2 | Math | 90 |
| 1 | Science | 85 |
| 2 | Science | 95 |

**Query**:  
Find the total marks per subject.

SELECT subject, SUM(marks) AS total\_marks

FROM grades

GROUP BY subject;

**Result**:

|  |  |
| --- | --- |
| **subject** | **total\_marks** |
| Math | 170 |
| Science | 180 |

**Why GROUP BY is Useful**:

* Helps aggregate data into meaningful summaries.
* Commonly used in reports and dashboards.

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

|  |  |  |
| --- | --- | --- |
| **Feature** | **GROUP BY** | **ORDER BY** |
| **Purpose** | Groups rows with the same column value into a single summary row. | Sorts the result set in ascending (ASC) or descending (DESC) order. |
| **Used With** | Aggregate functions (e.g., SUM, COUNT, AVG). | Any column, used for display order. |
| **Example** | Groups data by department to find total salaries. | Orders data by name or salary for better readability. |

**Example – GROUP BY:**

SELECT department, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department;

**Example – ORDER BY:**

SELECT name, salary

FROM employees

ORDER BY salary DESC;

**Key Point:**

* GROUP BY is used to aggregate data.
* ORDER BY is used to sort data.

**They can also be used together:**

SELECT department, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department

ORDER BY avg\_salary DESC;

This gives average salary per department sorted from highest to lowest.

**13. SQL Stored Procedure**

**Theory Questions:**

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

A stored procedure is a predefined group of SQL statements stored in the database, which can be executed repeatedly by calling its name.

**Standard SQL Query**:

* A single command executed directly (e.g., SELECT \* FROM students;).
* Runs only once unless written again.

**Stored Procedure**:

* A block of code stored in the database that can include multiple SQL statements, variables, control flow (IF, LOOP), and parameters.
* Can be executed many times by simply calling its name.

**Example – Stored Procedure Syntax** (in MySQL):

CREATE PROCEDURE GetAllStudents()

BEGIN

SELECT \* FROM students;

END;

To execute it:

CALL GetAllStudents();

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

|  |  |
| --- | --- |
| **Advantage** | **Explanation** |
| **Reusability** | Once created, the procedure can be called multiple times without rewriting the code. |
| **Performance Improvement** | Stored procedures are precompiled and stored in the database, so they execute faster than sending raw queries repeatedly. |
| **Security** | Allows controlled access to data. Users can execute the procedure without having direct access to underlying tables. |
| **Maintainability** | Centralized code makes it easier to maintain, update, or fix without changing client applications. |
| **Reduced Network Traffic** | Instead of sending multiple queries over the network, a single call executes multiple steps in the database. |
| **Parameter Support** | Stored procedures can accept parameters to customize behavior dynamically. |

**Example – Stored Procedure with Parameter**:

CREATE PROCEDURE GetStudentById(IN sid INT)

BEGIN

SELECT \* FROM students WHERE student\_id = sid;

END;

To execute it for student\_id = 1:

CALL GetStudentById(1);

**Summary**:

* Standard SQL query → Simple one-time command.
* Stored Procedure → Reusable, precompiled, and efficient group of commands stored in the database.

**14. SQL View**

**Theory Questions:**

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

🔹 A view in SQL is a virtual table based on the result of a SQL SELECT query.  
It does not store data physically but shows data from one or more tables as if it were a table.

**Difference between View and Table**:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Table** | **View** |
| **Data Storage** | Stores data physically in the database. | Does NOT store data physically (virtual). |
| **Purpose** | Used to store actual data. | Used to present data from one or more tables in a simplified or specific way. |
| **Data Modification** | You can insert, update, or delete (with restrictions). | Generally, read-only but can be updatable in some cases. |
| **Definition** | Created using CREATE TABLE. | Created using CREATE VIEW. |

**Example – Creating a View**:

CREATE VIEW StudentDetails AS

SELECT student\_id, name, age

FROM students

WHERE age >= 18;

**To use the view:**

SELECT \* FROM StudentDetails;

This shows all students aged 18 or older.

**2. Explain the advantages of using views in SQL databases.**

|  |  |
| --- | --- |
| **Advantage** | **Explanation** |
| **Simplifies Complex Queries** | Users can use the view instead of writing long complex joins and filters every time. |
| **Data Security** | Restricts user access to specific columns or rows by creating a view that shows only required data. |
| **Data Abstraction** | Users can work with a simple, consistent interface, while the underlying tables and joins are hidden. |
| **Reusability** | Once created, a view can be reused multiple times in queries. |
| **Consistency** | Provides a consistent data format or calculation (like pre-calculated averages) without recomputation. |

**Example – Security**:  
Instead of giving full access to the students table, create a view that hides sensitive columns like email.

CREATE VIEW PublicStudents AS

SELECT student\_id, name, age

FROM students;

Then, users can query:

SELECT \* FROM PublicStudents;

But they don’t see sensitive data.

**Summary**:

* A table stores data physically; a view shows data virtually based on a SELECT query.
* Views are helpful for security, simplicity, reusability, and abstraction.

**15. SQL Triggers**

**Theory Questions:**

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

A Trigger in SQL is a special stored procedure that automatically executes when a specific event occurs on a table (like INSERT, UPDATE, or DELETE).

**Purpose of Triggers**:

* Automatically perform actions to maintain data integrity.
* Perform automatic logging or validation.
* Enforce business rules.

**Types of Triggers**:

|  |  |
| --- | --- |
| **Type** | **When It Is Executed** |
| **BEFORE Trigger** | Executes before an action (INSERT, UPDATE, or DELETE) takes place. Used to validate or modify data before the operation happens. |
| **AFTER Trigger** | Executes after an action (INSERT, UPDATE, or DELETE) happens. Used for logging, updating related tables, etc. |
| **INSTEAD OF Trigger** | Used in views to perform an alternative action instead **of** the default operation. |

**Example Scenario**:

* Automatically log student additions to an audit table whenever a new student is inserted.

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

| **Trigger Type** | **Purpose** | **Example Use Case** |
| --- | --- | --- |
| ✅ **INSERT Trigger** | Fires when a new row is inserted into a table. | Automatically log new student details into an audit table whenever a new student is added. |
| ✅ **UPDATE Trigger** | Fires when an existing row is updated in a table. | Log previous and updated values of a record for tracking purposes. |
| ✅ **DELETE Trigger** | Fires when a row is deleted from a table. | Automatically archive deleted records into a separate table before deletion. |

👉 **Examples**:

**INSERT Trigger Example:**

CREATE TRIGGER after\_student\_insert

AFTER INSERT ON students

FOR EACH ROW

BEGIN

INSERT INTO audit\_log (action, student\_id, action\_time)

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

END;

* Whenever a new student is inserted, the trigger logs the action into audit\_log.

**UPDATE Trigger Example:**

CREATE TRIGGER before\_student\_update

BEFORE UPDATE ON students

FOR EACH ROW

BEGIN

INSERT INTO audit\_log (action, student\_id, old\_name, new\_name, action\_time)

VALUES ('UPDATE', OLD.student\_id, OLD.name, NEW.name, NOW());

END;

* Logs the old and new name of the student when their record is updated.

**DELETE Trigger Example**:

CREATE TRIGGER before\_student\_delete

BEFORE DELETE ON students

FOR EACH ROW

BEGIN

INSERT INTO deleted\_students (student\_id, name, deleted\_at)

VALUES (OLD.student\_id, OLD.name, NOW());

END;

* Stores deleted student records in deleted\_students table before deletion.

**Summary**:

* Triggers automate tasks.
* INSERT triggers run when a new row is added.
* UPDATE triggers run when an existing row is modified.
* DELETE triggers run when a row is removed.

**16. Introduction to PL/SQL**

**Theory Questions:**

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

PL/SQL (Procedural Language/SQL) is Oracle’s procedural extension of SQL.  
It allows you to write procedures, functions, loops, and control structures (like IF, FOR, WHILE) in addition to using normal SQL commands.

**Key Difference from SQL**:

* SQL is declarative: used to query and manipulate data.
* PL/SQL is procedural: adds programming constructs (variables, loops, conditions).

**Example – Simple PL/SQL Block**:

BEGIN

INSERT INTO students (student\_id, name, age) VALUES (1, 'Amit', 21);

COMMIT;

END;

It allows multiple operations to run in a single block with control flow.

**Why It Extends SQL**:

* Supports loops, conditions, exception handling, and variables.
* Enables complex logic, stored procedures, functions, and triggers.

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

|  |  |
| --- | --- |
| **Benefit** | **Explanation** |
| **Improved Performance** | Reduces network traffic by sending a block of statements in one go instead of individual SQL commands. |
| **Procedural Capabilities** | Allows use of variables, conditions, loops (FOR, WHILE), and exception handling, making complex business logic possible. |
| ✅ **Modularity** | Supports creating reusable procedures, functions, and packages, which improve code organization and maintainability. |
| ✅ **Error Handling** | Provides exception handling blocks to manage runtime errors gracefully. |
| ✅ **Security** | Can restrict direct access to tables by allowing access only through stored procedures. |
| ✅ **Maintainability** | Easier to manage large applications since logic is stored centrally in the database rather than application code. |
| ✅ **Data Integrity** | Helps implement complex validation rules before data is inserted or updated. |

**Simple Example of Conditional Logic in PL/SQL**:

DECLARE

v\_age INT;

BEGIN

v\_age := 20;

IF v\_age >= 18 THEN

INSERT INTO students (student\_id, name, age) VALUES (2, 'Rita', v\_age);

END IF;

COMMIT;

END;

**Summary**:

* PL/SQL adds procedural features to SQL.
* Useful for automation, error handling, reusable logic, and performance improvement.

**17. PL/SQL Control Structures**

**Theory Questions:**

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

Control structures in PL/SQL are constructs that allow you to control the flow of execution of code, making decisions or repeating actions.

**IF-THEN Control Structure**

Used to perform an action when a specific condition is true.

**Syntax**:

IF condition THEN

-- SQL statements

END IF;

Or with ELSE:

IF condition THEN

-- SQL statements when condition is true

ELSE

-- SQL statements when condition is false

END IF;

**Example**:

DECLARE

v\_age INT := 20;

BEGIN

IF v\_age >= 18 THEN

INSERT INTO students (student\_id, name, age) VALUES (1, 'Amit', v\_age);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Student is underage');

END IF;

COMMIT;

END;

**LOOP Control Structure**

Used to repeatedly execute a block of code.

**Syntax**:

LOOP

-- SQL statements

EXIT WHEN condition;

END LOOP;

**Example**:

DECLARE

counter INT := 1;

BEGIN

LOOP

INSERT INTO students (student\_id, name, age) VALUES (counter, 'Student\_' || counter, 18 + counter);

counter := counter + 1;

EXIT WHEN counter > 5;

END LOOP;

COMMIT;

END;

This inserts 5 student records.

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

|  |  |
| --- | --- |
| **Benefit** | **Explanation** |
| **Automation** | Enables automatic execution of multiple steps in sequence without manual intervention. |
| **Conditional Execution** | Allows execution of different actions based on dynamic data (e.g., IF age > 18, then insert). |
| **Repetition** | LOOP helps process repetitive tasks (e.g., inserting multiple records, processing batches of data). |
| **Exception Handling** | Handles errors and takes corrective actions to prevent crashes. |
| **Complex Business Logic** | Facilitates implementation of real-world business rules directly in the database. |

**Example Scenario**:

* Insert students only if they don’t already exist (using IF).
* Insert multiple students using LOOP.

**Summary**:

* IF-THEN: Makes decisions in PL/SQL.
* LOOP: Repeats actions.
* They help write powerful, efficient, and maintainable database logic.

**18. SQL Cursors**

**Theory Questions:**

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

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

In SQL, normally, queries return a complete result set all at once.  
But in PL/SQL, sometimes we need to process rows one by one — that’s where cursors are useful.

**Types of Cursors:**

|  |  |
| --- | --- |
| **Type** | **Description** |
| **Implicit Cursor** | Automatically created by PL/SQL when executing a SELECT, INSERT, UPDATE, or DELETE that returns only one row. No need to declare or open it explicitly. |
| **Explicit Cursor** | Manually declared by the programmer to handle multiple rows one by one. Gives more control over row-by-row processing. |

**Implicit Cursor Example**:

DECLARE

v\_student\_name students.name%TYPE;

BEGIN

SELECT name INTO v\_student\_name FROM students WHERE student\_id = 1;

DBMS\_OUTPUT.PUT\_LINE('Student Name: ' || v\_student\_name);

END;

No cursor declaration needed because it handles one row automatically.

**Explicit Cursor Example**:

DECLARE

CURSOR student\_cursor IS

SELECT student\_id, name FROM students;

v\_student\_id students.student\_id%TYPE;

v\_student\_name students.name%TYPE;

BEGIN

OPEN student\_cursor;

LOOP

FETCH student\_cursor INTO v\_student\_id, v\_student\_name;

EXIT WHEN student\_cursor%NOTFOUND;

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

END LOOP;

CLOSE student\_cursor;

END;

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

|  |  |
| --- | --- |
| **Scenario** | **Explanation** |
| **Multiple Rows Processing** | Use an explicit cursor when you need to process multiple rows one at a time (e.g., applying logic to each student record individually). |
| **More Control** | When you need fine control over opening, fetching, and closing the cursor. Useful for large datasets. |
| **Complex Logic** | If additional logic is needed per row, explicit cursors make it easy to implement inside the loop. |

**Example Use Case**:

* Updating all students’ records one by one and sending a message after each update → Use explicit cursor.

**Summary**:

* **Implicit Cursor**: For simple queries returning a single row.
* **Explicit Cursor**: When working with multiple rows, needing control over the fetch process.

**19. Rollback and Commit Savepoint**

**Theory Questions:**

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

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

In SQL, normally, queries return a complete result set all at once.  
But in PL/SQL, sometimes we need to process rows one by one — that’s where cursors are useful.

**Types of Cursors:**

|  |  |
| --- | --- |
| **Type** | **Description** |
| Implicit Cursor | Automatically created by PL/SQL when executing a SELECT, INSERT, UPDATE, or DELETE that returns only one row. No need to declare or open it explicitly. |
| Explicit Cursor | Manually declared by the programmer to handle multiple rows one by one. Gives more control over row-by-row processing. |

**Implicit Cursor Example**:

DECLARE

v\_student\_name students.name%TYPE;

BEGIN

SELECT name INTO v\_student\_name FROM students WHERE student\_id = 1;

DBMS\_OUTPUT.PUT\_LINE('Student Name: ' || v\_student\_name);

END;

No cursor declaration needed because it handles one row automatically.

**Explicit Cursor Example**:

DECLARE

CURSOR student\_cursor IS

SELECT student\_id, name FROM students;

v\_student\_id students.student\_id%TYPE;

v\_student\_name students.name%TYPE;

BEGIN

OPEN student\_cursor;

LOOP

FETCH student\_cursor INTO v\_student\_id, v\_student\_name;

EXIT WHEN student\_cursor%NOTFOUND;

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

END LOOP;

CLOSE student\_cursor;

END;

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

|  |  |
| --- | --- |
| **Scenario** | **Explanation** |
| **Multiple Rows Processing** | Use an explicit cursor when you need to process multiple rows one at a time (e.g., applying logic to each student record individually). |
| **More Control** | When you need fine control over opening, fetching, and closing the cursor. Useful for large datasets. |
| **Complex Logic** | If additional logic is needed per row, explicit cursors make it easy to implement inside the loop. |

**Example Use Case**:

* Updating all students’ records one by one and sending a message after each update → Use explicit cursor.

**Summary**:

* **Implicit Cursor**: For simple queries returning a single row.
* **Explicit Cursor**: When working with multiple rows, needing control over the fetch process.