Assignment

Module 5 – Introduction to SQL Database

(1) What is SQL, and why is it essential in database management?, Explain the difference between DBMS and RDBMS, Describe the role of SQL in managing relational databases, What are the key features of SQL?

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

SQL stands for *Structured Query Language*. It is a standard language that is used to work with databases. With the help of SQL, we can create a database, make tables, and also insert, update, delete or read data from those tables.

SQL is essential in database management because it gives a simple way to talk with the database. Instead of writing long programs, we can just write short SQL queries to get our work done. For example, if we want to see all students who got more than 60 marks, we can write one SQL query and get the result. This makes SQL very important for storing and managing data.

2. Explain the difference between DBMS and RDBMS.

- DBMS (Database Management System):

  DBMS is software that is used to store and manage data. In DBMS, data is mostly stored in files. It does not support relationships

between data. Data handling is slower and less secure. Example of DBMS is Microsoft Access or file system.

- RDBMS (Relational Database Management System):
  RDBMS is an advanced version of DBMS where data is stored in tables (rows and columns). It supports relationships between tables using keys like *primary key* and *foreign key*. It also provides better security, multi-user support, and faster performance. Example of RDBMS are MySQL, Oracle, SQL Server.

In short, DBMS is simple and used for small data, while RDBMS is powerful and used for large relational data.

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

In relational databases, data is stored in the form of tables. SQL plays the role of a language that allows us to interact with these tables. With SQL we can:

- Create tables and databases.
- Insert new records.
- Update existing records.
- Delete records.
- Retrieve or search data using queries.

Without SQL, it is very hard to manage relational databases because SQL provides a simple and standard way to do all operations. It also allows creating relationships between tables and ensures data security by giving permissions to users. That's why SQL is the backbone of relational databases.

4. What are the key features of SQL?

Some important features of SQL are:

1. Simple Language: SQL is easy to learn and understand, even for beginners.
2. Data Handling: It allows inserting, updating, deleting, and retrieving data from tables.
3. Data Definition: SQL can create and modify the structure of databases and tables.
4. Data Security: SQL provides permission and access control so that only authorized users can access data.
5. Portability: SQL works with almost all database systems like MySQL, Oracle, and SQL Server.
6. Scalability: SQL can handle small as well as very large amounts of data.
7. Standard Language: It is a standard language which is accepted globally for relational databases.

(2) What are the basic components of SQL syntax?, Write the general structure of an SQL SELECT statement, Explain the role of clauses in SQL statements.

1. What are the basic components of SQL syntax?

SQL syntax means the basic rules and structure we follow while writing SQL commands. Every SQL query has some important parts:

- Keywords: These are reserved words in SQL like SELECT, INSERT, UPDATE, DELETE, WHERE, etc. They tell what action to perform.
- Identifiers: Names given to database objects like tables, columns, or databases. Example: student, marks.
- Clauses: Parts of SQL commands that give extra instructions, like WHERE (for condition), ORDER BY (for sorting).
- Expressions: Used for calculations or logic, for example marks > 50.
- Operators: Symbols like =, <, >, AND, OR used in conditions.
- Statements: A complete SQL command, usually ending with a semicolon (;).

So, the basic components are keywords, identifiers, clauses, expressions, and operators.

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

The SELECT statement is the most common SQL command. It is used to fetch or display data from a table. Its general structure is:

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

- SELECT: Tells which columns of data we want to see.
- FROM: Tells from which table to take the data.
- WHERE: (Optional) Adds condition to filter rows.

- GROUP BY: (Optional) Groups rows having the same values.
- HAVING: (Optional) Adds condition on groups.
- ORDER BY: (Optional) Sorts the data in ascending or descending order.

Example:

```
SELECT name, marks
FROM student
WHERE marks > 50
ORDER BY marks DESC;
```

This query shows names and marks of students who scored more than 50, arranged in descending order.

3. Explain the role of clauses in SQL statements.

Clauses are important parts of SQL statements. They help in adding more details and conditions to a query. Some main roles of clauses are:

- WHERE clause: Used to filter records. Example: WHERE marks > 60 will only show students with marks greater than 60.
- ORDER BY clause: Used to sort the results in ascending or descending order.
- GROUP BY clause: Used to group rows that have the same values, like grouping students by class.
- HAVING clause: Used to put conditions on groups created by GROUP BY.

So, the role of clauses is to make SQL queries more powerful, specific, and meaningful. They allow us to filter, sort, and organize data as per the requirement.

(3) What are constraints in SQL? List and explain the different types of constraints, How do PRIMARY KEY and FOREIGN KEY constraints differ?, What is the role of NOT NULL and UNIQUE constraints?

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

Constraints in SQL are rules applied to columns of a table. They make sure that the data entered in the table is correct and follows certain conditions. Constraints help to maintain data accuracy and integrity.

Different types of constraints are:

- NOT NULL: Makes sure a column cannot have empty (NULL) values.
- UNIQUE: Ensures that all values in a column are different.
- PRIMARY KEY: A combination of NOT NULL and UNIQUE. It uniquely identifies each row in a table.
- FOREIGN KEY: Creates a link between two tables. It ensures that a value in one table matches a value in another table.

- CHECK: Ensures that all values in a column satisfy a specific condition. Example: age > 18.
- DEFAULT: Provides a default value for a column if no value is given.

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

- PRIMARY KEY:
  - It is used to uniquely identify each record in a table.
  - A table can have only one primary key.
  - It does not allow NULL or duplicate values.
  - Example: student_id in a student table.
- FOREIGN KEY:
  - It is used to link one table with another table.
  - A table can have many foreign keys.
  - It may allow duplicate values but those values must exist in the primary key of another table.
  - Example: course_id in an enrollment table linking to the course_id of course table.

So, primary key is used for unique identification inside a table, while foreign key is used for creating relationships between tables.

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

- NOT NULL:
    - This constraint makes sure that a column cannot have empty or missing values.
    - It is used when we always want a value in that column.
    - Example: In a student table, name column should be NOT NULL because every student must have a name.
- UNIQUE:
    - This constraint makes sure that all values in a column are different.
    - It prevents duplicate data in that column.
    - Example: In a student table, email column should be UNIQUE because no two students should have the same email.

So, NOT NULL ensures data must exist, while UNIQUE ensures no two rows have the same value.

(4) Define the SQL Data Definition Language (DDL), Explain the CREATE command and its syntax, What is the purpose of specifying data types and constraints during table creation?

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

DDL stands for *Data Definition Language*. It is a part of SQL that is used to define and manage the structure of a database. With DDL commands, we

can create, change, and delete database objects like tables, indexes, and views.

DDL does not deal with the actual data inside the tables, but only with the structure of the database.

Some common DDL commands are:

- CREATE → To create a new table or database.
- ALTER → To modify an existing table.
- DROP → To delete a table or database.
- TRUNCATE → To remove all records from a table but keep the structure.

So, DDL is mainly used for designing and organizing the structure of databases.

2. Explain the CREATE command and its syntax.

The CREATE command in SQL is used to make new objects like databases or tables. It is most commonly used to create tables where we store data.

General Syntax for creating a table:

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

- table_name: The name of the new table.
- column1, column2: Names of the columns in the table.

- datatype: The type of data each column will store (like INT, VARCHAR, DATE).
- constraint: Rules applied on the columns like NOT NULL, UNIQUE, PRIMARY KEY.

Example:

```
CREATE TABLE Student (
    student_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    age INT CHECK (age >= 18)
);
```

This will create a table named *Student* with four columns and some constraints.

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

When we create a table, it is important to define data types and constraints because:

- Data Types:
    - They define what kind of data a column can store (numbers, text, date, etc.).
    - Example: INT is used for numbers, VARCHAR for text, DATE for dates.
    - This ensures that only the correct type of data is entered in each column.

- Constraints:
    - They are rules that control the values in the columns.
    - Example: NOT NULL ensures a column cannot be empty, UNIQUE prevents duplicate values, and CHECK ensures values follow a condition.
    - This helps to maintain accuracy, reliability, and security of data.

In short: Data types make sure the column stores correct type of data, and constraints make sure the data is valid and meaningful.

(5) What is the use of the ALTER command in SQL?, How can you add, modify, and drop columns from a table using ALTER?

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

The ALTER command in SQL is used to change the structure of an existing table without deleting it.

With the help of ALTER, we can:

- Add new columns to a table.
- Modify the data type or size of existing columns.
- Rename a column or table (in some databases).
- Drop (remove) columns from a table.
- Add or remove constraints like PRIMARY KEY, FOREIGN KEY, UNIQUE, etc.

So, the main use of ALTER command is to update or change the design of a table after it has already been created.

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

(a) Add a new column:

We can add a new column to an existing table using ADD.

```
ALTER TABLE Student
ADD address VARCHAR(100);
```

This will add a new column named *address* to the Student table.

(b) Modify an existing column:

We can change the datatype or size of an existing column using MODIFY (or ALTER COLUMN in some databases).

```
ALTER TABLE Student
MODIFY name VARCHAR(80);
```

This changes the size of the *name* column from 50 to 80 characters.

(c) Drop a column:

We can remove a column from a table using DROP COLUMN.

```
ALTER TABLE Student
DROP COLUMN age;
```

This will delete the *age* column from the Student table.

(6) What is the function of the DROP command in SQL?, What are the implications of dropping a table from a database?

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

The DROP command in SQL is used to delete database objects permanently.

It can be used to drop a database, table, view, or index. Once an object is dropped, all the data stored in it is lost and cannot be recovered (unless there is a backup).

Example:

```
DROP TABLE Student;
```

This command will permanently delete the table *Student* and all the data stored in it.

So, the function of DROP is to completely remove unwanted database objects.

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

When we drop a table, some important things happen:

1. Data Loss: All records stored in the table are deleted permanently.
2. Structure Loss: The table design (columns, data types, constraints) is also removed.
3. Constraints Loss: Any keys (primary key, foreign key, unique) linked with that table are also lost.
4. Relationships Broken: If the dropped table was connected to other tables with foreign keys, those relationships will break.
5. No Rollback (in many cases): Once dropped, the table cannot be easily recovered unless there is a backup.

Example: If we drop a *Student* table, we lose student details permanently and also break links with other tables like *Courses* or *Marks*.

(7) Define the INSERT, UPDATE, and DELETE commands in SQL, What is the importance of the WHERE clause in UPDATE and DELETE operations?

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

- INSERT Command:

  The INSERT command is used to add new records (rows) into a table.

  Example:

- INSERT INTO Student (student_id, name, age)
- VALUES (1, 'Mukhtar', 20);

This will add a new student record into the *Student* table.

- UPDATE Command:

  The UPDATE command is used to change or modify the values of existing records in a table.

  Example:

- UPDATE Student
- SET age = 21
- WHERE student_id = 1;

This will change the age of the student with ID = 1 to 21.

- DELETE Command:

  The DELETE command is used to remove one or more records from

a table.

Example:

- DELETE FROM Student
- WHERE student_id = 1;

This will delete the record of the student whose ID = 1.

In short:

- INSERT → Add new data.
- UPDATE → Modify existing data.
- DELETE → Remove data.

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

The WHERE clause is very important when using UPDATE and DELETE because it specifies which rows should be updated or deleted.

- Without WHERE:
  - UPDATE will change all rows in the table.
  - DELETE will remove all rows in the table.
- With WHERE:
  - Only the rows that match the condition will be affected.

Example 1 – UPDATE without WHERE:

UPDATE Student SET age = 22;

This will set the age of *all students* to 22.

Example 2 – UPDATE with WHERE:

UPDATE Student SET age = 22 WHERE student_id = 2;

This will only change the age of the student whose ID = 2.

Example 3 – DELETE without WHERE:

DELETE FROM Student;

This will delete *all student records* (table becomes empty).

Example 4 – DELETE with WHERE:

DELETE FROM Student WHERE age < 18;

This will only delete students younger than 18.

(8) What is the SELECT statement, and how is it used to query data?, Explain the use of the ORDER BY and WHERE clauses in SQL queries

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

The SELECT statement is one of the most important commands in SQL. It is used to fetch (retrieve) data from one or more tables in a database.

General Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE condition

ORDER BY column_name;

- SELECT: Tells which columns of data we want.
- FROM: Tells from which table to get the data.
- WHERE: (Optional) Filters the records based on condition.
- ORDER BY: (Optional) Sorts the result.

Example:

SELECT name, age

FROM Student

WHERE age > 18;

This query will show the name and age of students whose age is greater than 18.

So, the SELECT statement is mainly used to ask questions (queries) from the database and get specific data as output.

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

- WHERE Clause:
    - The WHERE clause is used to filter records.
    - It makes sure only the rows that meet the condition are displayed.
    - Example:
    - SELECT * FROM Student
    - WHERE age > 20;

Shows only those students whose age is more than 20.

- ORDER BY Clause:
    - The ORDER BY clause is used to sort the result in ascending (ASC) or descending (DESC) order.
    - Example:
    - SELECT name, marks
    - FROM Student
    - ORDER BY marks DESC;

Shows student names and marks arranged from highest to lowest marks.

(9) What is the purpose of GRANT and REVOKE in SQL?, How do you manage privileges using these commands?

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

In SQL, GRANT and REVOKE are commands used for controlling access to the database. They are part of Data Control Language (DCL).

- GRANT: This command is used to give (grant) specific permissions to a user. Example: permission to read data, insert data, or update data.
- REVOKE: This command is used to take back (remove) permissions that were earlier given to a user.

Purpose:

They are used to manage security in databases so that only authorized users can access or change the data.

2. How do you manage privileges using these commands?

We manage privileges by giving and removing rights to users with the help of GRANT and REVOKE.

- Giving Privileges (GRANT):

  Example:
- GRANT SELECT, INSERT
- ON Student
- TO user1;

This allows *user1* to view (SELECT) and add (INSERT) data in the *Student* table.

- Removing Privileges (REVOKE):

  Example:

- REVOKE INSERT
- ON Student
- FROM user1;

This removes the permission of *user1* to insert data into the *Student* table.

(10) What is the purpose of the COMMIT and ROLLBACK commands in SQL?, Explain how transactions are managed in SQL databases.

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

In SQL, when we make changes to a table (like inserting, updating, or deleting data), those changes are done in a transaction.

To control these changes, we use COMMIT and ROLLBACK commands:

- COMMIT:
  - It is used to save all the changes permanently in the database.
  - Once we commit, the changes cannot be undone.
  - Example:
  - UPDATE Student SET age = 21 WHERE student_id = 1;
  - COMMIT;

The new age will be saved permanently.

- ROLLBACK:
  - It is used to undo the changes made in the current transaction.
  - If we made a mistake, rollback will return the database back to its previous state.

- Example:
- DELETE FROM Student WHERE age < 18;
- ROLLBACK;

The deletion will be canceled, and no records will be lost.

In short:

- COMMIT → Save changes permanently.
- ROLLBACK → Cancel changes and go back to the last saved state.

2. Explain how transactions are managed in SQL databases.

A transaction in SQL is a group of one or more SQL statements that are executed together. Transactions make sure that the database remains correct and reliable.

Transaction management follows the ACID properties:

1. Atomicity: Either all changes of a transaction happen, or none of them happen.
2. Consistency: A transaction keeps the database in a valid state before and after execution.
3. Isolation: Each transaction works independently, even if many users are working at the same time.
4. Durability: Once a transaction is committed, the changes are permanent, even if the system crashes.

Steps of Transaction Management:

- Start a transaction (automatically or using BEGIN).
- Perform SQL operations like INSERT, UPDATE, DELETE.
- If everything is correct → use COMMIT to save changes.

- If there is a mistake → use ROLLBACK to undo changes.

Example:

```
BEGIN;
INSERT INTO Student VALUES (5, 'Sana', 19);
UPDATE Student SET age = 20 WHERE student_id = 5;
COMMIT;
```

Both statements will be saved permanently together.

(11) Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?, How are joins used to combine data from multiple tables?

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

Concept of JOIN:

In SQL, a JOIN is used to combine rows from two or more tables based on a related column (usually a primary key in one table and a foreign key in another).

It allows us to see data that is stored in different tables together in one result.

Types of Joins:

1. INNER JOIN:
    - Returns only the rows that have matching values in both tables.
    - Non-matching rows are not included.

- Example: Show only students who are enrolled in a course.

2. LEFT JOIN (or LEFT OUTER JOIN):
   - Returns all rows from the left table and the matching rows from the right table.
   - If there is no match, NULL values are shown for the right table's columns.
   - Example: Show all students, even if they are not enrolled in a course.
3. RIGHT JOIN (or RIGHT OUTER JOIN):
   - Returns all rows from the right table and the matching rows from the left table.
   - If there is no match, NULL values are shown for the left table's columns.
   - Example: Show all courses, even if no student has enrolled in them.
4. FULL OUTER JOIN:
   - Returns all rows from both tables.
   - If there is no match, NULL values are shown on the side where no match is found.
   - Example: Show all students and all courses, whether they are related or not.

Summary Table:

JOIN Type	Result
INNER JOIN	Only matching rows from both tables
LEFT JOIN	All rows from left table + matching rows from right (else NULL)
RIGHT JOIN	All rows from right table + matching rows from left (else NULL)
FULL OUTER JOIN	All rows from both tables, matched or not (NULL where no match)

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

Joins are used when data is spread across multiple tables, and we need to see them together. Instead of storing all information in one big table, we divide it into smaller tables (normalization). Later, we use JOINs to combine them.

Example:

We have two tables:

- Student Table

  | student_id | name |

  |------------|--------|

  | 1 | Mukhtar |

  | 2 | Badi |

- Course Table

  | course_id | student_id | course_name |

|----------|------------|-------------|

| 101 | 1 | Database |

| 102 | 2 | Java |

Query using JOIN:

```
SELECT Student.name, Course.course_name
FROM Student
INNER JOIN Course
ON Student.student_id = Course.student_id;
```

Output:

name	course_name
Mukhtar	Database
Badi	Java

This shows how data from both tables is combined using a JOIN.

(12) What is the GROUP BY clause in SQL? How is it used with aggregate functions?, Explain the difference between GROUP BY and ORDER BY.

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 arrange rows that have the same values into groups.
- It is mostly used with aggregate functions (like COUNT, SUM, AVG, MAX, MIN) to perform calculations on each group.

Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
```

FROM table_name

GROUP BY column_name;

Example:

Suppose we have a *Sales* table:

salesman	amount
Mukhtar	2000
Danish	3000
Mukhtar	1500
Danish	2500

Query:

SELECT salesman, SUM(amount)

FROM Sales

GROUP BY salesman;

Output:

salesman	total_amount
Mukhtar	3500
Danish	5500

Here, GROUP BY arranged rows by *salesman*, and SUM calculated the total sales for each salesman.

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

Although both are SQL clauses, they are used for different purposes:

Feature	GROUP BY	ORDER BY
Purpose	Groups rows based on common column values	Sorts rows in ascending or descending order
Use With	Often used with aggregate functions (COUNT, SUM)	Used for arranging output neatly
Result	Returns one row for each group	Returns all rows, just sorted
Example	SELECT dept, COUNT(*) FROM Employee GROUP BY dept; → groups employees by department	SELECT name, age FROM Employee ORDER BY age DESC; → shows all employees sorted by age

In short:

- GROUP BY = Group rows to apply aggregate functions.
- ORDER BY = Sort rows in a specific order.

(13) What is a stored procedure in SQL, and how does it differ from a standard SQL query?, Explain the advantages of using stored procedures.

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

- A stored procedure in SQL is a set of pre-written SQL statements that are stored in the database and can be executed whenever needed.

- It is like a function in programming, where we write the code once and use it many times.

Syntax Example:

```
CREATE PROCEDURE GetAllStudents
AS
SELECT * FROM Student;
```

Difference from a standard SQL query:

- A standard SQL query is written and executed once. Example:
- SELECT * FROM Student;
- A stored procedure is saved in the database and can be executed many times using just its name. Example:
- EXEC GetAllStudents;

In short, stored procedures are reusable, faster, and more secure compared to writing queries again and again.

2. Explain the advantages of using stored procedures.

Stored procedures have many benefits:

1. Reusability:
    - We write the procedure once and call it whenever needed.
    - Saves time and reduces duplicate code.
2. Performance:
    - Stored procedures are pre-compiled by the database.
    - They run faster than normal queries.
3. Security:

- We can restrict direct access to tables and only allow users to run stored procedures.
- This keeps the database safe.

4. Maintainability:
    - If some logic needs to change, we only update the procedure, not every query.
5. Reduced Network Traffic:
    - Instead of sending multiple queries, we just call the procedure.
    - This reduces communication between application and database.

(14) What is a view in SQL, and how is it different from a table?, Explain the advantages of using viewsin SQL databases.

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

- A view in SQL is a virtual table that is created from the result of an SQL query.
- It does not store data itself, but shows data that comes from one or more tables.
- We can use a view just like a table for SELECT queries.

Example:

```
CREATE VIEW StudentView AS
SELECT student_id, name
FROM Student
WHERE grade = 'A';
```

This view will show only students with grade 'A'.

Difference between View and Table:

- Table: Stores the actual data physically in the database.
- View: Does not store data, it only displays data from one or more tables.
- Table: Can be updated directly.
- View: Usually read-only (though some views can be updatable).

In short: Table = real data storage, View = virtual window to see data.

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

Views provide many benefits in databases:

1. Simplicity:
    - Complicated queries can be saved as a view.
    - Users can just use the view instead of writing long queries again.
2. Security:
    - Views can hide sensitive columns (like salary, password).
    - Only selected data is shown to the user.
3. Data Independence:
    - Even if the underlying table changes, the view can remain the same.
    - This protects applications from changes in the database structure.
4. Reusability:
    - Once created, a view can be used multiple times in different queries.

5. Readability:
    - Views make queries more readable and clear.

(15) What is a trigger in SQL? Describe its types and when they are used, Explain the difference between INSERT, UPDATE, and DELETE triggers.

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

- A trigger in SQL is a special type of stored program that runs automatically when an event happens in a table.
- The event can be INSERT, UPDATE, or DELETE.
- Triggers are useful for maintaining data consistency, security, and automatic actions.

Example:

If we want to automatically keep a log whenever a student record is deleted, we can use a trigger.

Types of Triggers in SQL:

1. BEFORE Trigger
    - Runs before the action (INSERT, UPDATE, DELETE).
    - Used when we want to check or modify values before saving them in the table.
2. AFTER Trigger
    - Runs after the action has happened.
    - Used when we want to take an action after data is changed, like updating logs or sending notifications.
3. INSTEAD OF Trigger (mainly in views)

- Used when we want to perform a different action instead of the normal INSERT, UPDATE, or DELETE.
- Useful for views, because views normally don't allow direct changes.

When are triggers used?

- For automatic auditing (keeping logs of changes).
- For data validation before saving records.
- For enforcing business rules (like not allowing negative salary).
- For maintaining referential integrity between tables.

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

1. INSERT Trigger
   - Runs when a new row is inserted into the table.
   - Example use: Automatically add current date or created_by details when a new record is inserted.
2. UPDATE Trigger
   - Runs when an existing row is updated in the table.
   - Example use: Keep track of old values before they are changed.
3. DELETE Trigger
   - Runs when a row is deleted from the table.
   - Example use: Store deleted records in a backup table for safety.

(16) What is PL/SQL, and how does it extend SQL's capabilities?, List and explain the benefits of using PL/SQL.

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

- PL/SQL (Procedural Language/SQL) is an extension of SQL developed by Oracle.
- SQL is mainly used for querying and managing data, but it cannot handle complex programming logic like loops, conditions, or functions.
- PL/SQL adds programming features such as variables, loops, conditions, functions, and error handling to SQL.

In simple words:

- SQL = only for data handling.
- PL/SQL = SQL + programming power.

How PL/SQL extends SQL's capabilities:

- It allows writing blocks of code (with BEGIN and END).
- It supports IF...ELSE, FOR loop, and other control statements.
- It supports procedures, functions, and triggers.
- It provides exception handling to manage runtime errors.

So, PL/SQL makes SQL more powerful and flexible for real applications.

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

1. Better Performance

- PL/SQL allows sending a block of code (many SQL statements) at once to the database, which reduces network traffic and improves speed.

2. Modularity (Reusable Code)
   - Code can be stored in the form of procedures and functions, which can be reused in different applications.
3. Error Handling
   - PL/SQL provides exception handling to catch and handle runtime errors, which makes programs safer.
4. Supports Complex Logic
   - Unlike SQL, PL/SQL supports loops, conditions, and decision-making, which helps in building powerful programs.
5. Security
   - PL/SQL programs can hide the business logic from users by only providing access to stored procedures/functions instead of tables directly.
6. Integration
   - PL/SQL can easily integrate with other programming languages and applications, making it flexible to use.

(17) What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures, How do control structures in PL/SQL help in writing complex queries?

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

Control structures in PL/SQL are commands that control the flow of execution in a program.

They allow us to decide when a statement should run and how many times it should run.

Without control structures, PL/SQL would only run statements one by one in sequence.

The main types are:

- Conditional control (IF statements)
- Iterative control (LOOP statements)
- Sequential control (GOTO, etc.)

(a) IF-THEN structure:

- It is used for decision making.
- If a given condition is true, then the statements inside IF block are executed.

Syntax:

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

Example:

```
IF marks >= 40 THEN
   DBMS_OUTPUT.PUT_LINE('Pass');
END IF;
```

(b) LOOP structure:

- LOOP is used when we want to repeat a set of statements multiple times.
- A LOOP runs again and again until we use the EXIT condition.

Syntax:

```
LOOP
   -- statements
   EXIT WHEN condition;
END LOOP;
```

Example:

```
x := 1;
LOOP
   DBMS_OUTPUT.PUT_LINE(x);
   x := x + 1;
   EXIT WHEN x > 5;
END LOOP;
```

(This will print numbers 1 to 5)

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

Control structures make PL/SQL more powerful than normal SQL because:

1. Decision Making
    - Using IF-THEN, we can execute different queries depending on conditions.
    - Example: Update salary only if employee is permanent.
2. Repetition

- Using LOOP, FOR, or WHILE, we can repeat tasks (like inserting multiple records or checking multiple conditions).

3. Error Handling
   - With control structures, we can check conditions before running a query, which avoids errors.
4. Complex Logic
   - They allow combining SQL queries with programming logic (like conditions, counters, iterations), which normal SQL cannot do.

In short: Control structures make PL/SQL behave like a programming language, so we can write complex, flexible, and efficient database programs.

(18) What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors, When would you use an explicit cursor over an implicit one?

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

A cursor in PL/SQL is a temporary area in memory which stores the result of an SQL query.

When we run a SELECT statement, PL/SQL uses a cursor to keep track of rows returned by the query.

Cursors are mainly used to process query results row by row.

There are two types of cursors:

(a) Implicit Cursor

- Created automatically by PL/SQL when we run a simple SQL statement (like INSERT, UPDATE, DELETE, or SELECT INTO).
- We don't need to declare it.
- It is managed internally by PL/SQL.
- Example:

UPDATE employees SET salary = salary + 1000 WHERE dept_id = 10;

Here, PL/SQL automatically creates an implicit cursor.

(b) Explicit Cursor

- Created manually by the programmer when the query returns multiple rows.
- We must declare, open, fetch, and close it step by step.
- It gives more control to process each row one by one.

Steps for Explicit Cursor:

1. Declare the cursor
2. Open the cursor
3. Fetch rows from it
4. Close the cursor

Example:

```
CURSOR c1 IS SELECT name, salary FROM employees;
OPEN c1;
FETCH c1 INTO v_name, v_salary;
CLOSE c1;
```

Difference between Implicit and Explicit Cursor:

Implicit Cursor	Explicit Cursor
Created automatically	Created by programmer
Used for single-row queries	Used for multi-row queries
No need of OPEN, FETCH, CLOSE	Needs OPEN, FETCH, CLOSE
Less control	More control

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

We use an explicit cursor when:

1. The query returns multiple rows and we want to process each row one by one.

   Example: Display all employee names with their salary.
2. We need more control over fetching data (like skipping some rows or stopping in the middle).
3. When we want to perform different actions on each row from the result.

In short:

- Use implicit cursor → for single-row queries.
- Use explicit cursor → for multi-row queries where row-by-row processing is required.

(19) Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?, When is it useful to use savepoints in a database transaction?

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

A SAVEPOINT in SQL is a temporary marker within a transaction.

It allows us to save the state of the database at a certain point so that we can roll back (undo) only part of the transaction instead of the whole one.

Key points:

- A transaction can have multiple savepoints.
- If something goes wrong, we can use ROLLBACK TO savepoint_name to undo only the changes done after that savepoint.
- The command COMMIT will save all changes permanently, including those before and after savepoints.

Example:

```
START TRANSACTION;

UPDATE accounts SET balance = balance - 500 WHERE id = 1;
SAVEPOINT sp1;

UPDATE accounts SET balance = balance + 500 WHERE id = 2;
ROLLBACK TO sp1;  -- Undo the second update only

COMMIT;  -- Save the first update permanently
```

In this example, the first update is saved, but the second one is undone using ROLLBACK TO sp1.

Interaction:

- SAVEPOINT + ROLLBACK → undo part of the transaction.
- SAVEPOINT + COMMIT → once committed, savepoints are no longer available (transaction ends).

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

Savepoints are useful when:

1. A big transaction has many steps, and we want to undo only a specific step if an error happens.
2. While testing or debugging, to check partial results without losing all work.
3. In banking or money transfer systems, where multiple updates happen but we may need to undo just one faulty step.
4. When different parts of a program update the database, and only one part fails.

In short:

Savepoints are useful to control transactions more efficiently, giving us the option to undo only part of the work instead of everything.