1. Q1

   Database recovery is the process of restoring a database to a correct, consistent state after a failure. The purpose of database recovery is to safeguard the data integrity and allow the system to return to normal operations with minimal loss of data. Failures can include system crashes, disk failures, or software errors, which disrupt the normal functioning of the database.

   Recovery Techniques

   There are two main recovery techniques used in database management:

1. **Roll Forward**:
   - This technique involves reapplying changes recorded in the redo logs to bring the database to the most recent consistent state. Redo logs store the actions of committed transactions, so in case of failure, roll forward ensures that all completed transactions are reflected in the database.
2. **Rollback**:
   - Rollback undoes the changes made by uncommitted or incomplete transactions. By using information in the undo logs, this technique removes any partial or erroneous changes, bringing the database back to its last consistent state prior to failure.

   These recovery techniques ensure data integrity and consistency, which are essential in database management.

2. Q2

   An index is a database structure that improves the speed of data retrieval operations on a database table. It works by creating a data structure that allows the database to locate records quickly without scanning the entire table. Indexes are crucial for enhancing query performance, especially on large databases.

   Types of Indexes:

1. **Single-Column Index**: An index based on a single column in a table.
2. **Composite Index**: An index created on multiple columns to enhance query performance for searches involving those columns.
3. **Unique Index**: Ensures that all values in the indexed column(s) are unique, often used to enforce primary or unique key constraints.
4. **Clustered Index**: Physically reorders the rows in the table to match the index, allowing faster data retrieval but limited to one per table.
5. **Non-Clustered Index**: Keeps the table rows in the original order and uses a separate structure for the index; multiple non-clustered indexes can exist per table.

   Joins

   Joins are operations that retrieve data from multiple tables based on a related column between them. Joins are essential in relational databases for combining data across tables.

   Types of Joins:

1. **Inner Join**: Returns only the rows where there is a match in both tables.
2. **Left Outer Join**: Returns all rows from the left table and matching rows from the right table; if no match, NULL values are returned for columns from the right table.
3. **Right Outer Join**: Returns all rows from the right table and matching rows from the left table; if no match, NULL values are returned for columns from the left table.
4. **Full Outer Join**: Returns all rows when there is a match in one of the tables, with NULLs where there is no match.
5. **Cross Join**: Combines all rows from the left table with all rows from the right table, resulting in a Cartesian product.

Indexes and joins are fundamental for optimizing data retrieval and enhancing relational database performance.

3. Q3

PL/SQL (Procedural Language/Structured Query Language) is structured into blocks that allow for better organization of code, making it modular and manageable. A PL/SQL block is made up of four main sections:

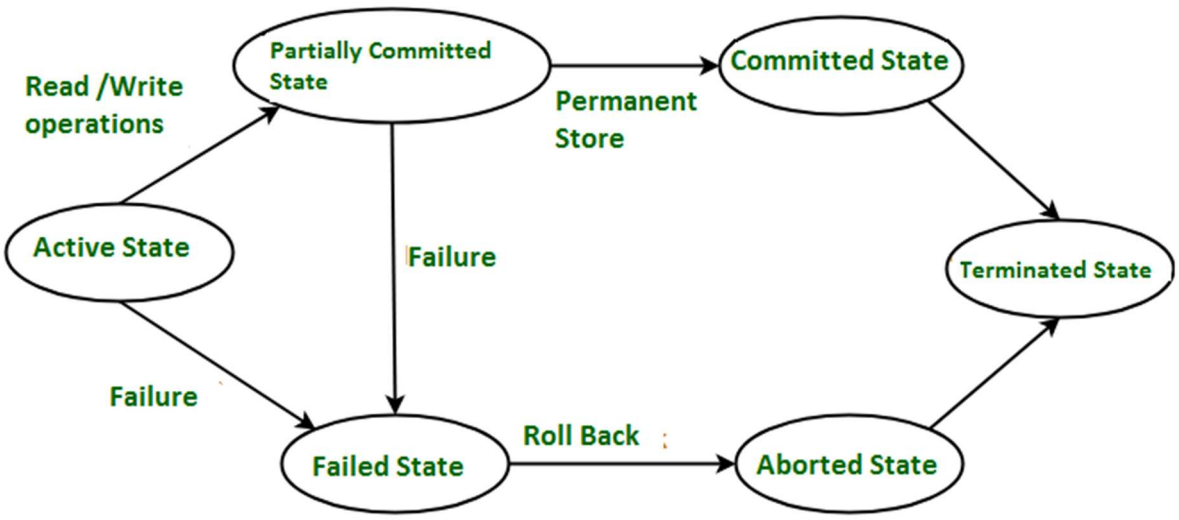
- **Declaration Section**: This is optional and begins with the DECLARE keyword. Here, variables, constants, cursors, and other elements required in the program are defined. This section prepares resources and allows the PL/SQL block to operate with specific data values.
- **Execution Section**: This is the only mandatory section of a PL/SQL block and begins with the BEGIN keyword. It includes the logic and instructions of the program, such as SQL statements, conditional statements, and loops. This is the core part of the PL/SQL block, as it carries out the main task of the program.
- **Exception Handling Section**: This optional section begins with the EXCEPTION keyword and is used to manage errors that may arise during execution. If an error occurs, the PL/SQL block will move to this section to handle exceptions and execute recovery or error-handling code, maintaining the program's stability.
- **End**: A PL/SQL block always ends with the END keyword, which signals the end of the block's code. This mandatory statement helps the program flow and differentiates individual blocks within the program.

4. Q4

Functions in PL/SQL offer various advantages that make coding efficient and manageable:

- **Code Reusability**: Once a function is created, it can be called multiple times throughout a program or by other programs. This eliminates code duplication and allows the same logic to be reused without rewriting the code, saving time and reducing errors.
- **Modularity**: Functions allow complex problems to be broken down into simpler, manageable parts. By creating multiple functions for different tasks, the program becomes more organized and easier to debug, maintain, and understand. Modularity also enhances teamwork, as different programmers can work on different functions independently.

5. Q5



Transaction States in DBMS

A transaction in DBMS refers to a single unit of work that is performed on a database, and it must meet specific **ACID** properties to maintain data integrity:

- **Atomicity**: Ensures that all operations within a transaction are completed fully, or none are applied. This property protects the database from incomplete transactions.
- **Consistency**: Guarantees that each transaction brings the database from one valid state to another, maintaining the integrity constraints set on the data.
- **Isolation**: Ensures that transactions execute independently of one another. This prevents data inconsistency caused by concurrent transactions.
- **Durability**: Ensures that once a transaction is committed, it remains permanently in the database, even if the system crashes.

Transition State Diagram:

- **Active**: The transaction is being executed.
- **Partially Committed**: The transaction has completed its operations but hasn't yet committed.
- **Committed**: The transaction has successfully completed, and changes are saved.
- **Failed**: A transaction error has occurred, prompting a rollback.
- **Aborted**: The transaction is rolled back, and any changes made are undone.

6. Q6

WHERE: Used to filter records based on a specified condition. It limits the rows returned by a query.

Example: **SELECT * FROM Employee WHERE salary > 50000;**

This query retrieves records of employees with a salary greater than 50,000.

GROUP BY: Groups rows sharing a common attribute and is often used with aggregate functions (SUM, COUNT, etc.) to summarize data.

Example: **SELECT deptno, COUNT(empno) FROM Employee GROUP BY deptno;**

This query counts the number of employees in each department by grouping based on **deptno**

7. Q7
   UPPER(): Converts a string to uppercase.

   Syntax: **UPPER(string)**
   Example: **SELECT UPPER(name) FROM Employee;** – Converts the **name** field to uppercase.

   LOWER(): Converts a string to lowercase.
   Syntax: **LOWER(string)**
   Example: **SELECT LOWER(name) FROM Employee;** – Converts the **name** field to lowercase.

   SUBSTR(): Extracts a substring from a string, based on a specified starting point and length.

   Syntax: **SUBSTR(string, start_position, length)**
   Example: **SELECT SUBSTR(name, 1, 3) FROM Employee;** – Extracts the first three characters from **name.**

   LENGTH(): Returns the length of a string.
   Syntax: **LENGTH(string)**
   Example: **SELECT LENGTH(name) FROM Employee;** – Returns the length of the **name** field.

   CONCAT(): Concatenates two strings into one.
   Syntax: **CONCAT(string1, string2)**
   Example: **SELECT CONCAT(first_name, last_name) FROM Employee;** – Concatenates **first_name** and **last_name**.

8. Q8
   A view is a virtual table that is based on the result set of a SELECT query. Here's how to create a view with attributes ACC_no and PAN from the Depositor table for customers with balances over 100,000.

   // CODE
   CREATE VIEW HighValueDepositors AS
   SELECT ACC_no, PAN
   FROM Depositor
   WHERE Balance > 100000;

9. Q9
   Sequence Creation: A sequence is an object in the database that generates unique numbers sequentially, which can be used for auto-generating values for primary keys.
   // Code
   CREATE SEQUENCE Seq_1
   START WITH 1
   INCREMENT BY 1
   MINVALUE 1
   MAXVALUE 20;

Using Sequence to Insert Values: Here's how to use Seq_1 to insert values into the Student table, where Seq_1.NEXTVAL provides a unique ID:
//code
INSERT INTO Student (ID, Name)
VALUES (Seq_1.NEXTVAL, 'John Doe');

10. Q10
    Construct a stored procedure to calculate and update the annual bonus for employees based on performance ratings and salary.

    A stored procedure can be created to automate the calculation of employee bonuses based on specific criteria.

    //code:

```
CREATE PROCEDURE CalculateBonus
AS
BEGIN
  FOR emp IN (SELECT empno, salary, rating FROM Employee)
  LOOP
    UPDATE Employee
    SET bonus = salary * CASE
                  WHEN emp.rating = 'A' THEN 0.10
                  WHEN emp.rating = 'B' THEN 0.05
                  ELSE 0.02
                END
    WHERE empno = emp.empno;
  END LOOP;
END;
```

11. Q11

- **GRANT**: This command provides specific permissions to users or roles on database objects such as tables, views, or procedures.
  - **Syntax**: GRANT privilege ON object TO user;
  - **Example**: GRANT SELECT, INSERT ON Employee TO User1; – Grants User1 permission to select and insert records in the Employee table.
- **REVOKE**: This command removes previously granted permissions from users or roles.
  - **Syntax**: REVOKE privilege ON object FROM user;
  - **Example**: REVOKE INSERT ON Employee FROM User1; – Removes the insert permission on the Employee table from User1.

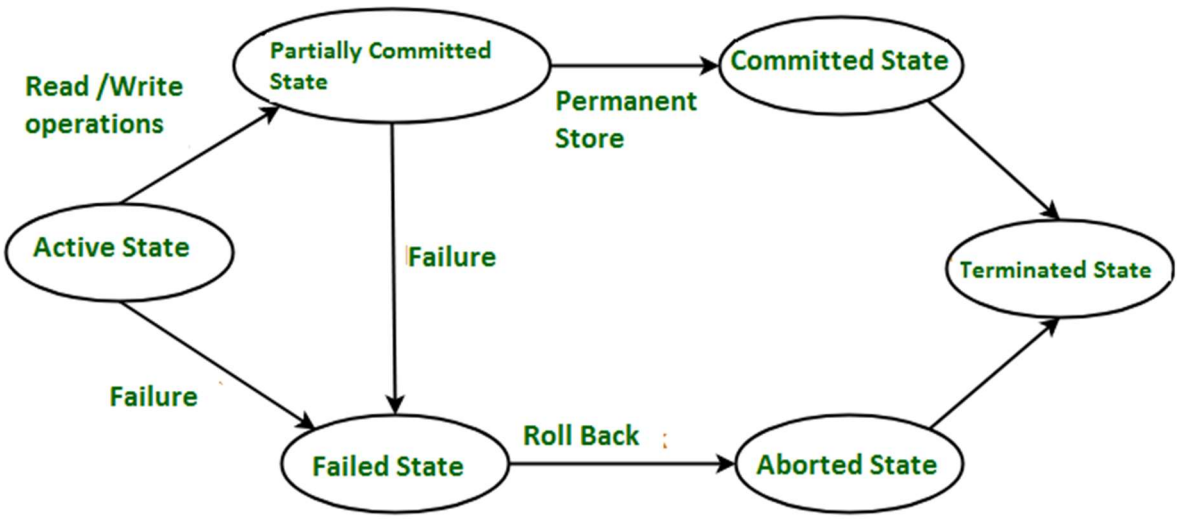
Grant and revoke commands are essential for maintaining database security and controlling access levels for different users.

12. Q12

In DBMS, users are created and managed by administrators to ensure secure access to database resources. Privileges can also be assigned or revoked to control what users can and cannot do.

- **Creating a User**:
  - **Syntax**: CREATE USER username IDENTIFIED BY password;
  - **Example**: CREATE USER User1 IDENTIFIED BY pass123;
  - This command creates a new user User1 with the password pass123.
- **Deleting a User**:
  - **Syntax**: DROP USER username;
  - **Example**: DROP USER User1;
  - This removes the user User1 from the database entirely, along with any privileges they were assigned.
- **Assigning Privileges**:
  - **Syntax**: GRANT privilege TO username;
  - **Example**: GRANT SELECT, INSERT ON Employee TO User1;
  - This command grants User1 the ability to SELECT and INSERT data in the Employee table. Privileges allow administrators to control user actions, such as which tables they can access and what operations they can perform on them.

13. Q13



Transaction States in DBMS

The transaction state diagram illustrates the states a transaction can go through during its lifecycle, reflecting the **ACID** properties (Atomicity, Consistency, Isolation, and Durability):

- **Active State**: This is the initial state where a transaction is currently being executed.
- **Partially Committed**: After all statements within the transaction have been executed, the transaction reaches this state but is not yet fully committed.
- **Committed**: This state is reached once the transaction is completed and all changes are permanently saved to the database.
- **Failed**: If an error occurs during the transaction, it moves to this state, requiring a rollback to undo any partial changes made.
- **Aborted**: If a transaction fails, it can be rolled back, returning to the state before the transaction started.

  The diagram for transaction states typically shows these transitions, ensuring database consistency at each stage.

14. Q14

    IF Statement in PL/SQL: The IF statement is a control structure used for decision-making, executing specific code blocks based on conditions. It allows for conditional logic in PL/SQL blocks.

    //Code:

```
IF condition THEN
  -- statements to execute if condition is true
ELSIF another_condition THEN
  -- statements for another condition
ELSE
  -- statements if all conditions are false
END IF;
```

    example code:

```
DECLARE
  salary NUMBER := 50000;
BEGIN
  IF salary > 40000 THEN
    DBMS_OUTPUT.PUT_LINE('High Salary');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Average Salary');
  END IF;
END;
```

15. Q15

    Explain Cursor and Trigger. Write the syntax to create, open, fetch, and close a cursor.

- **Cursor**: A cursor is a pointer used in PL/SQL to handle query results row by row, making it ideal for operations on individual rows.
    - **Types**: Implicit cursors (handled automatically by PL/SQL) and explicit cursors (defined by the user).

Syntax to Create, Open, Fetch, and Close a Cursor:

```
DECLARE
  CURSOR cursor_name IS SELECT * FROM table_name;
  record_name table_name%ROWTYPE;
BEGIN
  OPEN cursor_name;
  FETCH cursor_name INTO record_name;
  CLOSE cursor_name;
END;
```

Trigger: A trigger is a stored procedure that automatically executes in response to certain events on a table, such as insertions, updates, or deletions.

- **Example Syntax**:

```
//CODE
CREATE OR REPLACE TRIGGER trigger_name
BEFORE INSERT ON table_name
FOR EACH ROW
BEGIN
  -- Trigger logic
END;
```

16. Q16

Exception handling in PL/SQL manages runtime errors, allowing the program to continue or gracefully terminate. There are **predefined exceptions** (for common errors like division by zero) and **user-defined exceptions**.

Example of Exception Handling:

```
//Code:
   DECLARE
  salary NUMBER := 0;
BEGIN
  salary := 100 / salary;  -- This will cause a divide-by-zero error
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Division by zero error.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('An unexpected error occurred.');
END;
```

17. Q17.

List DDL, DML, DCL, and TCL commands.

- **DDL (Data Definition Language)**: Commands that define or alter database structures.

    - Examples: CREATE, ALTER, DROP, TRUNCATE
- **DML (Data Manipulation Language)**: Commands that manipulate data within tables.
    - Examples: SELECT, INSERT, UPDATE, DELETE
- **DCL (Data Control Language)**: Commands that control access to data.
    - Examples: GRANT, REVOKE
- **TCL (Transaction Control Language)**: Commands that manage transaction states.
    - Examples: COMMIT, ROLLBACK, SAVEPOINT

  These commands allow full control over database structure, data management, security, and transactional integrity.

18. Q18

    In databases, triggers are special types of stored procedures that automatically execute in response to specific events or actions on a table or view. There are several types of triggers, which can be categorized based on the kind of event they respond to. Common types of triggers include:

    1. DML (Data Manipulation Language) Triggers
    2. DDL (Data Definition Language) Triggers
    3. Logon Triggers
    4. Logoff Triggers
    5. *Compound Triggers*
    6. *INSTEAD OF Triggers*
    7. *System Triggers*

    ### 1. *DDL (Data Definition Language) Triggers*

    DDL triggers are fired when certain DDL events (such as creating, altering, or dropping tables, views, or other schema objects) occur in the database. These triggers allow administrators to capture and respond to structural changes in the database schema.

    Key Features of DDL Triggers:

    - They respond to events such as CREATE, ALTER, and DROP operations on schema objects.
    - Useful for auditing changes to the database schema, enforcing security policies, or preventing unauthorized changes.
    - Can be used to log DDL operations, prevent the modification or deletion of critical objects, or enforce naming conventions**.**

    Example Use Case:

    - Preventing users from dropping certain tables or views by raising an error or logging the event whenever a DROP statement is executed.

    ### 2. *DML (Data Manipulation Language) Triggers*

DML triggers are the most commonly used type of triggers, and they respond to actions like INSERT, UPDATE, or DELETE on database tables.

Key Features of DML Triggers:
- They can be set to execute *before* or *after* an insert, update, or delete operation.
- Often used for enforcing business rules, auditing, and maintaining data integrity.
- DML triggers can be defined on a per-row or per-statement basis, meaning they can be triggered for each affected row or only once for the entire operation.

Example Use Case:
- *Before Insert Trigger:* Automatically calculating and setting the value of a column (e.g., setting a default value before a new row is inserted).
- *After Update Trigger:* Logging the old and new values of a record after an update, often used for auditing changes made to the data.

3. *Logon Triggers*
Logon triggers are fired when a user session is established in the database. They can be used for purposes like enforcing security policies, logging user activity, or restricting access based on certain conditions.

Key Features of Logon Triggers:
- They execute when a user successfully logs into the database.
- Can be used to implement security measures (e.g., tracking which users are accessing the system) or restrict user behavior (e.g., limiting access during off-hours).
- Often used to set session-level parameters or execute custom checks when a user connects.

Example Use Case:
- Restricting access to certain database features or tables based on the user's role, or logging login attempts for auditing purposes.

These triggers serve different needs in database management, from enforcing business logic to maintaining system security and integrity.