UNIT 1 ADVANCED SQL

1. Controlling Program Flow, Conditional Statements, and Loops:

Conditional Statements:

- **IF statement: Used to execute code based on a condition.**

Syntax:

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

- **CASE: Conditional expression that returns values based on conditions.**

Syntax:

```
SELECT CASE
      WHEN condition THEN result
      ELSE other_result
     END
FROM table;
```

- **Loops:**

- **WHILE loop: Repeats the block of code as long as the condition is true.**

Syntax:

```
WHILE condition DO
   -- SQL statements
END WHILE;
```

- **FOR loop: Iterates over a range of values or a set of results.**

Syntax:

```
FOR counter IN start_value..end_value LOOP
   -- SQL statements
END LOOP;
```

2. Views:

- **Definition: A view is a virtual table based on the result of an SQL query. It doesn't store data but shows data from one or more tables.**

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2 FROM table_name WHERE condition;
```

- **Usage: Simplify complex queries, provide security by restricting access to certain data, and abstract the database schema for easier management.**

3. Stored Functions:

- **Definition: A stored function is a set of SQL statements that can be stored and executed repeatedly.**

Syntax:

```
CREATE FUNCTION function_name (parameter1 data_type, ...)
RETURNS return_data_type
AS
BEGIN
   -- SQL statements
   RETURN value;
END;
```

- **Usage: Perform calculations, return specific results, and encapsulate business logic for reusability.**

4. Cursors:

- **Definition: A cursor allows row-by-row processing of query results. It is typically used when you need to process each row individually.**

- **Types:**

1. **Implicit Cursor: Automatically created when you execute a SELECT statement.**
2. **Explicit Cursor: Defined by the user for more control over the rows.**

Syntax:

```
DECLARE cursor_name CURSOR FOR

SELECT column1, column2 FROM table;

OPEN cursor_name;

FETCH NEXT FROM cursor_name INTO @var1, @var2;

CLOSE cursor_name;

DEALLOCATE cursor_name;
```

- **Usage: Useful for operations requiring multiple steps for each row of data.**

5. Triggers:

- **Definition: A trigger is a stored procedure that is automatically executed in response to certain events on a table or view.**

- **Types:**

1. **BEFORE Trigger: Executes before the event (INSERT, UPDATE, DELETE) on the table.**
2. **AFTER Trigger: Executes after the event.**

Syntax:

```
CREATE TRIGGER trigger_name

BEFORE INSERT ON table_name
```

FOR EACH ROW

BEGIN

-- SQL statements

END;

- **Usage: Enforce business rules, maintain referential integrity, and automatically update or audit data.**

UNIT 2 TRANSACTIONS AND CONCURRENCY CONTROL MECHANISM

1. Transaction:

- **Definition: A transaction is a sequence of one or more SQL operations (like INSERT, UPDATE, DELETE) that are executed as a single unit of work. A transaction is treated as a single, indivisible unit that either completes successfully or is rolled back.**

- **Example: A bank transfer transaction involving multiple steps like debit from one account and credit to another.**

2. Properties of a Transaction (ACID Properties):

1. **Atomicity: The transaction is all or nothing. If any part of the transaction fails, the entire transaction is rolled back.**

2. **Consistency: The database must always remain in a consistent state before and after the transaction. Any transaction must take the database from one valid state to another.**

3. **Isolation: Each transaction is executed as if it is the only transaction in the system. It should not be affected by other concurrent transactions.**

4. **Durability: Once a transaction has been committed, its changes are permanent and will survive system crashes.**

3. State of a Transaction:

A transaction goes through various states during its lifecycle:

- **New: The transaction has been started but not yet executed.**
- **Active: The transaction is currently being executed.**
- **Partially Committed: The transaction has executed all its operations but has not yet been committed to the database.**
- **Committed: The transaction has successfully completed and its changes have been permanently applied.**
- **Aborted: The transaction has failed and has been rolled back. All changes made by the transaction are undone.**
- **Failed: The transaction has encountered an error and cannot proceed, waiting for rollback or recovery.**

4. Executing Transactions Concurrently:

- **Definition: Concurrent execution refers to multiple transactions being executed at the same time or overlapping in time.**

- **Problem in Concurrent Execution:**

i. **Lost Update: Two transactions modify the same data simultaneously without proper synchronization, leading to data loss.**
ii. **Temporary Inconsistency (Dirty Read): A transaction reads data that is being modified by another transaction but hasn't yet been committed, leading to inconsistency.**
iii. **Uncommitted Data (Rollback): A transaction reads data from another transaction that was rolled back.**
iv. **Deadlock: Two or more transactions are blocked, each waiting for the other to release resources, causing a standstill.**

- **Solution to Issues:**

i. **Locking Mechanisms: Ensuring transactions acquire locks on the data they are modifying to prevent conflicts.**
ii. **Isolation Levels: Control the visibility of changes made by one transaction to other concurrent transactions.**

5. Schedules:

- **Definition: A schedule is an ordered sequence of operations (transactions' actions like READ, WRITE) from multiple transactions.**

- **Types of Schedules:**

- **Serial Schedule: Transactions are executed one after another without overlapping. It is guaranteed to be conflict-free.**
- **Non-Serial Schedule: Transactions are executed concurrently, with interleaved operations. This can lead to problems if not managed properly.**
- **Conflict Serializable: A schedule is conflict-serializable if its operations can be rearranged (without changing the order of operations within a transaction) into a serial schedule.**
- **View Serializable: A schedule is view-serializable if it preserves the same final state as some serial schedule, even if the order of operations is different.**

6. Concurrency Control Mechanisms:

- **Locking:**

i. **Shared Lock: Allows multiple transactions to read the same data but not modify it.**
ii. **Exclusive Lock: Prevents any other transactions from reading or modifying the data.**
iii. **Deadlock Detection: Periodically checking for cycles in the resource allocation graph to detect deadlocks.**
iv. **Timestamp Ordering: Each transaction is assigned a unique timestamp, and transactions are ordered based on these timestamps to ensure consistency.**

- **Two-Phase Locking (2PL):**

i. **Growing Phase: A transaction can acquire locks but cannot release any locks.**

ii. **Shrinking Phase: A transaction can release locks but cannot acquire new ones.**

2PL guarantees conflict serializability, but can lead to deadlocks.

- **Optimistic Concurrency Control: Transactions execute without acquiring locks but must validate that no conflicts occurred before committing.**

- **Multiversion Concurrency Control (MVCC): Allows multiple versions of data to exist so that transactions can work with a snapshot of data without locking.**

UNIT 3 DATABASE INTEGRITY AND SECURITY CONCEPTS

1. Domain Constraints:

- **Definition: Domain constraints specify the valid values that a column can hold in a database. They define the type, format, and range of data for each attribute (column).**

- **Types of Domain Constraints:**

- **Data Type: The type of data allowed (e.g., INT, VARCHAR, DATE).**
- **Range: Limits the data to a certain range (e.g., AGE must be between 18 and 100).**
- **Format: Ensures data follows a specific format (e.g., email format).**
- **Not Null: A constraint that ensures the column cannot have NULL values.**

- **Example:**

```
CREATE TABLE employees (
  emp_id INT,
  emp_name VARCHAR(100),
  emp_salary DECIMAL(10, 2) CHECK (emp_salary > 0)
```

);

2. Referential Integrity:

- **Definition: Referential integrity ensures that relationships between tables are maintained consistently. Specifically, it guarantees that foreign keys in a table must either be NULL**

or correspond to a valid primary key in another table.

- **Foreign Key: A field in one table that uniquely identifies a row in another table.**

- **Key Points:**

  - **ON DELETE CASCADE: Deletes rows in the referencing table when the referenced row is deleted.**
  - **ON UPDATE CASCADE: Updates rows in the referencing table when the referenced row is updated.**
  - **ON DELETE SET NULL: Sets the foreign key to NULL when the referenced row is deleted.**

  - **Example:**

```
CREATE TABLE orders (
  order_id INT PRIMARY KEY,
  customer_id INT,
  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
  ON DELETE CASCADE
);
```

3. Database Security Concepts:

- **Definition: Ensures that data in a database is protected from unauthorized access, modification, or destruction.**

- **Types of Security:**

- **Authentication: Verifying the identity of users (e.g., username and password).**
- **Authorization: Granting specific privileges to authenticated users (e.g., read, write, delete access).**
- **Encryption: Encrypting data to protect it during storage and transmission.**
- **Auditing: Monitoring and logging database access and changes for security purposes.**

Unit 4: Crash Recovery

1. Failure Classification:

- **Definition: Database systems are vulnerable to various types of failures that can result in data loss or corruption. Failure classification helps to understand the type of failure and the necessary recovery methods.**

- **Types of Failures:**

i. **Transaction Failures: Occurs when a transaction cannot complete due to reasons such as invalid operations, violations of constraints, or resource issues.**
ii. **System Failures: A failure that affects the entire database management system (DBMS), such as a crash, power loss, or hardware malfunction.**
iii. **Media Failures: Involves the loss of data due to physical damage to storage media (e.g., hard drives).**

- **Failure Causes:**

i. **Logical Failures: Occur due to errors in the transaction (e.g., violation of constraints, incorrect queries).**
ii. **Physical Failures: Occur when hardware or software crashes, causing loss of system state or data.**

- **Recovery Process:**

i. **Undo: Rolling back a transaction to its state before execution.**
ii. **Redo: Reapplying changes made by committed transactions that were not written to disk before the crash.**
iii. **Write-Ahead Logging (WAL): Ensures that logs are written to disk before changes to the actual database. This is used to ensure recovery after a crash.**

2.Crash Recovery Techniques:

1) **Log-Based Recovery: Uses a log of all changes made by transactions to the database to undo or redo operations in case of failure.**
2) **Write-Ahead Logging (WAL): Writes the log entries to disk before the actual database changes are made. This helps ensure durability during recovery.**
3) **Checkpointing: Periodically saving the database state to a stable storage. After a crash, the database can be recovered from the last checkpoint.**
4) **Shadow Paging: Instead of overwriting data pages, a new version is created and pointers are updated. In case of failure, the original pages can be used for recovery.**

3. Recovery Manager:

- **The recovery manager ensures that after a crash, the database can be brought back to a consistent state. It uses logs (WAL) to ensure that the database can either redo or undo changes made by transactions.**

- **Two-Phase Commit Protocol (2PC): Ensures that all distributed transactions across multiple databases either commit or rollback, ensuring atomicity and consistency.**