DBMS Mid 2 QnA

1. a. Explain in detail about 1NF, 2NF and 3NF with suitable examples.

First Normal Form (1NF):
A table is in 1NF if:

- It only contains atomic (indivisible) values.
- Each column contains values of a single type.
- Each column must have a unique name.
- The order in which data is stored does not matter.

Example:
Consider a table with the following structure:

StudentID	Name	Subjects
1	John Doe	Math, Physics
2	Jane Doe	Chemistry

This table is not in 1NF because the `Subjects` column contains multiple values. To convert it to 1NF, each subject should be in a separate row.

StudentID	Name	Subject
1	John Doe	Math
1	John Doe	Physics
2	Jane Doe	Chemistry

Second Normal Form (2NF):
A table is in 2NF if:

- It is in 1NF.
- It has no partial dependencies; i.e., non-key attributes are fully dependent on the primary key.

Example:
Consider the 1NF table:

StudentID	Name	Subject	Teacher
1	John Doe	Math	Mr. Smith
1	John Doe	Physics	Mrs. Johnson

StudentID	Name	Subject	Teacher
2	Jane Doe	Chemistry	Dr. Brown

Here, `Name` depends only on `StudentID` but not on `Subject`. To bring this table to 2NF, separate `Student` and `Subject` details.

StudentID	Name
1	John Doe
2	Jane Doe

StudentID	Subject	Teacher
1	Math	Mr. Smith
1	Physics	Mrs. Johnson
2	Chemistry	Dr. Brown

Third Normal Form (3NF):
A table is in 3NF if:

- It is in 2NF.
- It has no transitive dependencies; i.e., non-key attributes are not dependent on other non-key attributes.

Example:
Consider the 2NF tables:

StudentID	Name
1	John Doe
2	Jane Doe

StudentID	Subject	TeacherID
1	Math	101
1	Physics	102
2	Chemistry	103

TeacherID	Teacher
101	Mr. Smith
102	Mrs. Johnson
103	Dr. Brown

Here, `Teacher` is dependent on `TeacherID`, not directly on `StudentID` or `Subject`. By creating a separate `Teacher` table, we eliminate the transitive dependency, thus achieving 3NF.

1. b. Explain in detail about Triggers and Active databases.

Triggers:
A trigger is a stored procedure in a database that automatically invokes or executes in response to certain events on a particular table or view. Common events include `INSERT`, `UPDATE`, and `DELETE`.

Example:
Creating a trigger to log changes in the `Students` table:

```
CREATE TABLE StudentLog (
    LogID INT PRIMARY KEY,
    StudentID INT,
    Action VARCHAR(50),
    ActionTime DATETIME
);

CREATE TRIGGER LogStudentChanges
AFTER INSERT ON Students
FOR EACH ROW
BEGIN
    INSERT INTO StudentLog (StudentID, Action, ActionTime)
    VALUES (NEW.StudentID, 'INSERT', NOW());
END;
```

This trigger logs every insertion into the `Students` table into the `StudentLog` table.

Active Databases:
Active databases are databases that include triggers, constraints, and other mechanisms to enforce business rules and data integrity automatically. They react to changes in the database without explicit request from the user.

Features of Active Databases:

- **Triggers**: Automatically execute specified actions in response to certain events.
- **Constraints**: Enforce rules at the database level (e.g., primary keys, foreign keys, checks).
- **Stored Procedures**: Precompiled collections of SQL statements that can be executed as a program.

Advantages:

- Enforce data integrity and business rules consistently.
- Reduce application code complexity by moving logic to the database.
- Improve performance by reducing round-trips between application and database.

2. a. What is Transaction? Explain the life cycle of the transaction with a neat diagram.

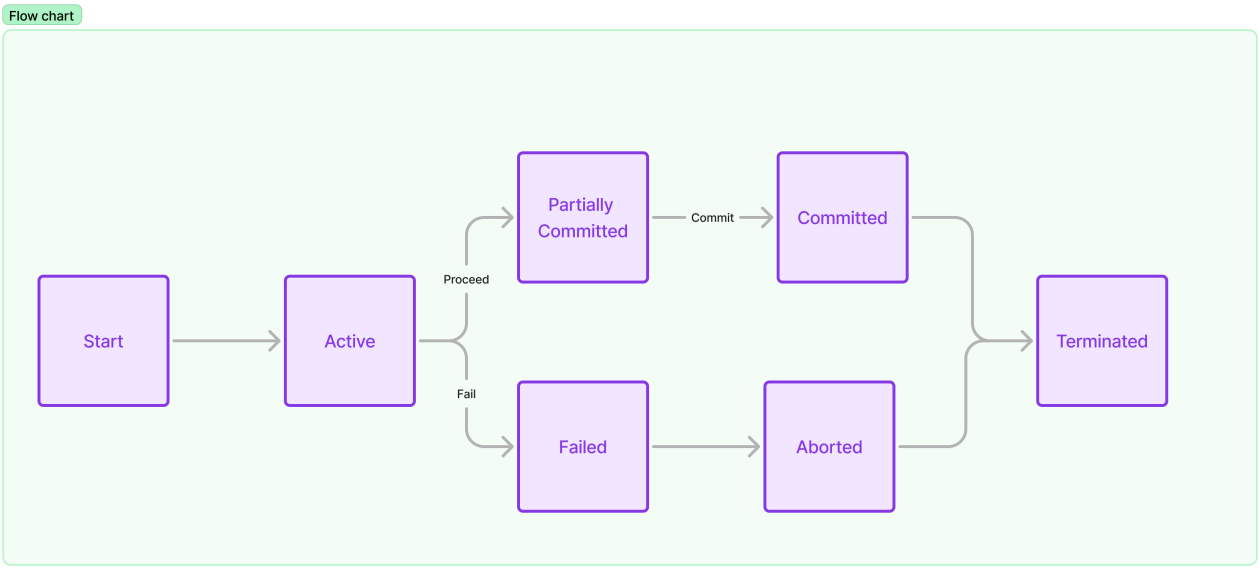
What is a Transaction?

A transaction is a unit of work that consists of a sequence of one or more operations performed on a database. Transactions ensure data integrity and consistency by adhering to the ACID properties: Atomicity, Consistency, Isolation, and Durability.

Life Cycle of a Transaction:

1. **Active:** The transaction is being executed.
2. **Partially Committed:** All operations have been executed, and the transaction is waiting for a commit.
3. **Committed:** The transaction has been successfully completed, and changes are made permanent.
4. **Failed:** The transaction cannot proceed due to an error.
5. **Aborted:** The transaction has been rolled back, and changes are undone.
6. **Terminated:** The transaction has ended (either committed or aborted).

Diagram:



2. b. Discuss about transaction recovery techniques.

Transaction recovery techniques are essential for maintaining the integrity and consistency of a database in the event of a failure. They ensure that the database can recover to a consistent state.

Types of Failures:

1. **Transaction Failures:** Logical errors or system crashes.

2. **System Failures:** Hardware or software failures.
3. **Media Failures:** Disk crashes or data corruption.

Recovery Techniques:

1. **Log-Based Recovery:**

   - **Write-Ahead Logging (WAL):** Before any changes are made to the database, the changes are first recorded in a log.
   - **Undo Logging:** Logs before values (old values) to undo changes if a transaction fails.
   - **Redo Logging:** Logs after values (new values) to redo changes if a transaction commits.

   Example:

   ```
   T1: Start
   T1: Write X (X=100 -> X=150)
   Log: <T1, X, 100>
   T1: Commit
   ```

2. **Checkpointing:**

   - Periodically save the current state of the database and the transaction log to disk.
   - During recovery, the system can start from the last checkpoint, reducing the amount of log data to process.

   Example:

   ```
   [Checkpoint] T1, T2 active
   ```

3. **Shadow Paging:**

   - Maintain two versions of the database: the current page table and the shadow page table.
   - Changes are made to the current page table. If the transaction commits, the shadow page table is updated.

   Example:

   ```
   Current Page Table: P1, P2, P3
   Shadow Page Table:  S1, S2, S3
   ```

4. **Deferred Update:**

- All changes are kept in a buffer until the transaction reaches the commit point.
- Once committed, the changes are written to the database.

Example:

```
Buffer: Write X=150
Commit: Apply changes to database
```

5. **Immediate Update:**
    - Changes are applied immediately to the database but are logged so they can be undone if necessary.

Example:

```
Write X=150 (log the old value X=100)
```

3. What is serializability? Explain Conflict serializability with an example.

Serializability:

Serializability is a concept in database concurrency control that ensures that a schedule (sequence of operations from multiple transactions) results in a consistent database state. A schedule is serializable if it produces the same result as some serial execution of the transactions.

Conflict Serializability:

Conflict serializability is a type of serializability that ensures a schedule can be transformed into a serial schedule by swapping non-conflicting operations.

Conflict:
Two operations conflict if they:

1. Belong to different transactions.
2. Access the same data item.
3. At least one of the operations is a write.

Example:

Consider two transactions T1 and T2:

- T1: Read(A), Write(A)

- T2: Read(A), Write(A)

Non-Serializable Schedule:

```
T1: Read(A)
T2: Read(A)
T2: Write(A)
T1: Write(A)
```

Conflict Pairs:

- `T1 Read(A)` and `T2 Write(A)`
- `T2 Read(A)` and `T1 Write(A)`

This schedule is not serializable because there is no way to swap the operations to achieve a serial order without changing the final outcome.

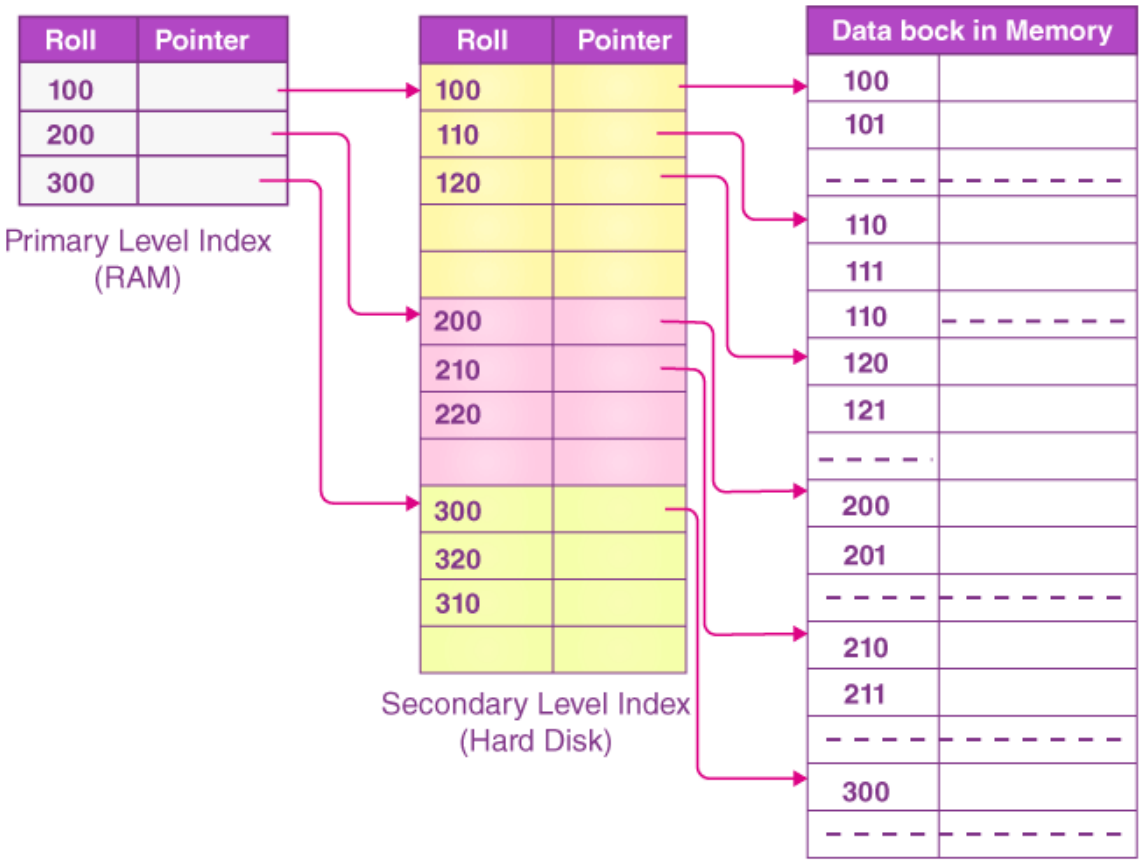
Serializable Schedule:

```
T1: Read(A)
T1: Write(A)
T2: Read(A)
T2: Write(A)
```

The operations are in a serial order, and the final result will be consistent with a serial execution of T1 and T2.

4. a. Define Indexing and explain about types of indexing.

Definition:
Indexing is a data structure technique that improves the speed of data retrieval operations on a database table. An index is a copy of selected columns of data from a table that is designed to speed up retrieval.



Types of Indexing:

1. **Primary Index:**

   - Built on the primary key of the table.
   - Each index entry points to the data block containing the actual record.
   - One-to-one mapping between index entries and data records.

   Example: Index on the `EmployeeID` column in an employee table.
2. **Secondary Index:**

   - Built on non-primary key columns.
   - There can be multiple secondary indices on a table.
   - One-to-many mapping between index entries and data records.

   Example: Index on the `LastName` column in an employee table.
3. **Clustered Index:**

   - The order of the rows in the table corresponds to the order of the rows in the index.
   - Only one clustered index per table since it defines the physical storage order.

   Example: Index on the `OrderDate` column in an orders table.
4. **Non-clustered Index:**

- Separate from the actual data rows.
- Can have multiple non-clustered indices on a table.
- Index entries contain pointers to the data rows.

Example: Index on the `Email` column in a user table.

5. **Composite Index:**

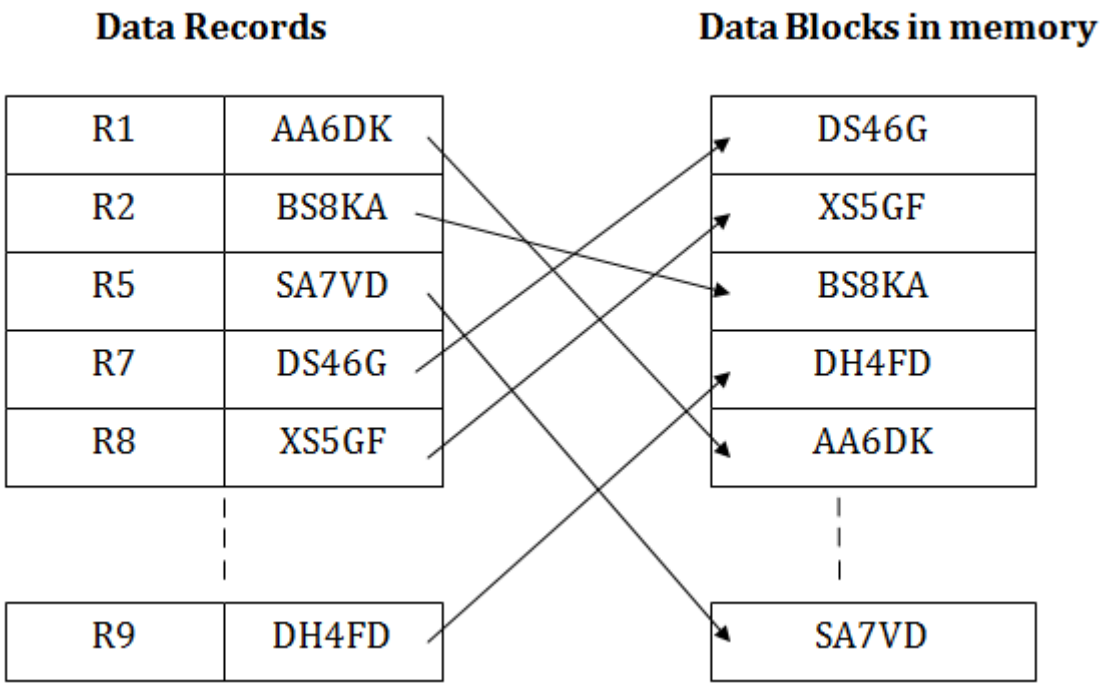
- Built on multiple columns of a table.
- Useful for queries that filter on more than one column.

Example: Index on the `FirstName` and `LastName` columns in a contacts table.

4. b. Explain about Indexed sequential access method (ISAM).

What is ISAM?

Indexed Sequential Access Method (ISAM) is a file organization method that combines the benefits of both sequential and direct access. It allows for efficient searching, insertion, deletion, and updates of records in a file by using an index to quickly locate data.



Characteristics of ISAM:

1. **Primary Index:** A primary index is built on the primary key of the file.
2. **Overflow Areas:** When a data block (bucket) is full, additional records are stored in an overflow area.
3. **Sequential Access:** Records can be accessed sequentially using pointers.
4. **Static Structure:** The structure of ISAM is relatively static, making it suitable for files with infrequent updates.

Structure of ISAM:

1. **Index Levels:**
   - **Primary Index:** Points to data blocks (buckets).
   - **Secondary Index (optional):** May be used for faster access.
2. **Data Blocks (Buckets):** Store the actual records.
3. **Overflow Blocks:** Store records that cannot fit into their designated data blocks.

Operations in ISAM:

1. **Search Operation:**

   - Use the primary index to find the appropriate data block.
   - Search within the data block for the desired record.
   - If the record is not found, follow pointers to overflow blocks if they exist.

   Example:

   ```
   Search for key 75:
   - Use the primary index to locate the data block.
   - Search the data block for key 75.
   - If not found, check overflow blocks.
   ```

2. **Insertion Operation:**

   - Locate the appropriate data block using the primary index.
   - Insert the record into the data block if space is available.
   - If the data block is full, add the record to the overflow block.
   - Update pointers as necessary.

   Example:

   ```
   Insert key 60:
   - Locate the data block using the primary index.
   - Insert 60 into the data block.
   - If the data block is full, insert 60 into an overflow block.
   - Update pointers.
   ```

3. **Deletion Operation:**

   - Locate the record using the primary index and data block.
   - Remove the record from the data block or overflow block.
   - If necessary, reorganize the data block or overflow block to maintain efficiency.

   Example:

```
Delete key 45:
- Locate key 45 using the primary index.
- Remove key 45 from the data block or overflow block.
- Reorganize blocks if needed.
```

4. **Update Operation:**

   - Locate the record using the primary index and data block.
   - Update the record in the data block or overflow block.
   - If the update causes the record to be moved, handle as an insertion and deletion.

   Example:

```
Update key 80 to 85:
- Locate key 80 using the primary index.
- Update the record to 85.
- If the data block changes, handle the update as an insertion and
deletion.
```

Advantages of ISAM:

1. Efficient for both sequential and random access.
2. Suitable for read-mostly workloads with infrequent updates.
3. Simple and straightforward implementation.

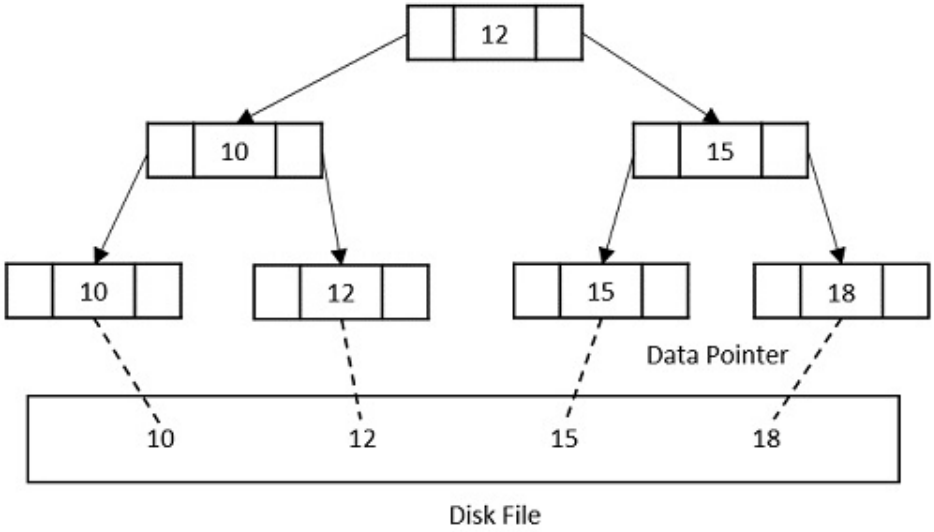
Disadvantages of ISAM:

1. Static structure makes handling frequent updates challenging.
2. Overflow areas can become fragmented over time, reducing efficiency.
3. Requires periodic reorganization to maintain performance.

5. a. What is a B+ tree? Explain in detail the operations of B+ trees.

What is a B+ Tree?

A B+ Tree is a type of self-balancing tree data structure that maintains sorted data and allows for efficient insertion, deletion, and search operations. It is an extension of the B-Tree and is commonly used in databases and file systems. Unlike B-Trees, all values in a B+ Tree are stored at the leaf level, and internal nodes only store keys to guide the search.



Characteristics of B+ Trees:

1. **Balanced Tree Structure:** All leaf nodes are at the same level.
2. **Ordered:** Keys are maintained in a sorted order.
3. **Nodes:** Internal nodes store keys and pointers, while leaf nodes store keys and values.
4. **Linked Leaves:** Leaf nodes are linked together, facilitating range queries.
5. **Degree:** Each node can have a minimum and maximum number of children, defined by the tree's degree (m).

Operations on B+ Trees:

1. **Search Operation:**

   - Begin at the root node and compare the search key with the keys in the node.
   - Follow the appropriate pointer (child) based on the comparison.
   - Repeat the process until a leaf node is reached.
   - Search the leaf node for the key.

   Example:

   ```
   Search for key 45:
   - Start at root: Compare 45 with keys.
   - Move to appropriate child node.
   - Continue until reaching the leaf.
   - Locate key 45 in the leaf node.
   ```

2. **Insertion Operation:**

   - Start at the root and find the appropriate leaf node where the key should be inserted.

- Insert the key in the leaf node in the correct sorted order.
- If the leaf node overflows (exceeds the maximum allowed keys), split the leaf node.
  - Insert the middle key into the parent node.
  - If the parent node overflows, repeat the splitting process up the tree.
- Adjust pointers to maintain the structure.

Example:

```
Insert key 25:
- Locate the leaf node for insertion.
- Insert 25 in the correct order.
- If the leaf node overflows, split and promote the middle key to the
parent.
- Repeat if necessary up to the root.
```

3. **Deletion Operation:**

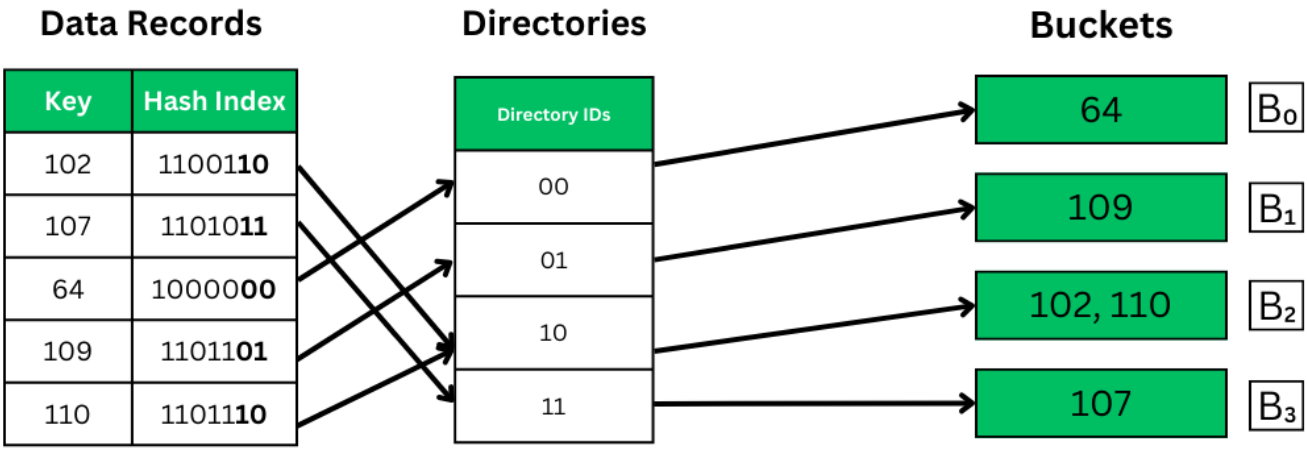
- Locate the key to be deleted in the leaf node.
- Remove the key from the leaf node.
- If the leaf node underflows (has fewer than the minimum allowed keys), rebalance by:
  - Redistributing keys from adjacent siblings.
  - Merging with a sibling if redistribution is not possible.
- Adjust pointers and keys in parent nodes as necessary.

Example:

```
Delete key 30:
- Locate key 30 in the leaf node.
- Remove key 30.
- If the leaf node underflows, redistribute keys with siblings or merge
nodes.
- Update parent nodes as needed.
```

5. b. What is Hashing? Explain in detail about hashing techniques.

Definition:
Hashing is a technique used to map data of arbitrary size to fixed-size values (hash values or hash codes). In databases, hashing is used to quickly locate a data record given its search key.



Hashing Techniques:

1. **Static Hashing:**
    - The number of primary pages (buckets) is fixed.
    - A hash function maps keys to a fixed number of buckets.
    - Can lead to overflow pages when a bucket gets full (handled by overflow chaining).

    Example: A hash table with 10 buckets using a hash function `h(key) = key % 10`.
2. **Dynamic Hashing:**
    - The number of buckets can grow or shrink dynamically.
    - Common techniques include Extendible Hashing and Linear Hashing.

    Extendible Hashing:
    - Uses a directory to map hash values to buckets.
    - The directory can grow in size, and buckets can be split as needed.
    - Example: Initially, two buckets with a hash function based on the first bit of the key. As buckets fill up, more bits are used.

    Linear Hashing:
    - Buckets are split in a linear order.
    - Uses a level indicator to decide when to split buckets.
    - Example: Initially, one bucket with a hash function `h0(key) = key % 1`. As records increase, split the bucket and use `h1(key) = key % 2`.
3. **Open Addressing:**
    - All elements are stored in the hash table itself.
    - When a collision occurs, it probes for the next empty slot.
    - Techniques include Linear Probing, Quadratic Probing, and Double Hashing.

    Linear Probing:

- If a collision occurs at `h(key)`, check the next slot `h(key) + 1`, `h(key) + 2`, etc.
- Example: For a key that hashes to position 3, if position 3 is occupied, check positions 4, 5, etc.

Quadratic Probing:

- Probes at intervals of `1^2, 2^2, 3^2, ...` slots.
- Example: For a key that hashes to position 3, if position 3 is occupied, check positions `3 + 1^2`, `3 + 2^2`, `3 + 3^2`, etc.

Double Hashing:

- Uses two hash functions.
- If a collision occurs at `h1(key)`, the next position is determined by `h2(key)`.
- Example: First hash `h1(key) = key % 10`, second hash `h2(key) = 1 + (key % 9)`.

6. a. What is a lock in DBMS? Discuss different types of locks in DBMS.

What is a Lock in DBMS?

A lock is a mechanism used to control concurrent access to data in a database. Locks prevent conflicts when multiple transactions try to access the same data simultaneously, ensuring data integrity and consistency.

Types of Locks:

1. **Binary Locks:**

   - **Lock:** The data item is locked.
   - **Unlock:** The data item is unlocked.

   Example:

   ```
   Lock(X)
   Unlock(X)
   ```

2. **Shared and Exclusive Locks:**

   - **Shared Lock (S):** Multiple transactions can read the data item.
   - **Exclusive Lock (X):** Only one transaction can write to the data item.

   Example:

```
S(X)  // Shared lock on X
X(X)  // Exclusive lock on X
```

3. **Intention Locks:**

   - **Intention Shared (IS):** Intent to acquire a shared lock on a lower-level node.
   - **Intention Exclusive (IX):** Intent to acquire an exclusive lock on a lower-level node.
   - **Shared Intention Exclusive (SIX):** Shared lock on the node with intention exclusive locks on lower-level nodes.

   Example:

```
IS(DB)  // Intent to acquire shared lock on a table within DB
IX(DB)  // Intent to acquire exclusive lock on a table within DB
```

4. **Degree of Locking (Granularity):**

   - Locks can be applied at various levels of granularity: Database, Table, Page, Row, etc.

   Example:

```
Database Lock (DB)
Table Lock (T)
Page Lock (P)
Row Lock (R)
```

5. **Two-Phase Locking Protocol (2PL):**

   - **Growing Phase:** Transaction acquires all necessary locks.
   - **Shrinking Phase:** Transaction releases locks and cannot acquire new ones.

   Example:

```
Growing Phase: Lock(X), Lock(Y)
Shrinking Phase: Unlock(X), Unlock(Y)
```

6. **Deadlock and Deadlock Resolution:**

   - **Deadlock:** Occurs when two or more transactions wait for each other to release locks.
   - **Resolution Techniques:**
     - **Timeouts:** Abort transactions that have been waiting for too long.

- **Deadlock Detection and Recovery:** Periodically check for deadlocks and abort one or more transactions to resolve them.

Example:

```
Transaction T1 locks X, waits for Y.
Transaction T2 locks Y, waits for X.
```

6. b. Explain multiple granularity of locking protocol with example.

Multiple granularity locking allows transactions to lock objects at different levels of granularity, such as the entire database, a table, a page, or a row. This protocol helps in improving the performance by allowing different transactions to operate on different levels of data.

Types of Locks:

- **Intention Locks:** Used to indicate the intention to acquire more granular locks.
  - `IS (Intention Shared)` : Intention to acquire shared locks on finer granularity.
  - `IX (Intention Exclusive)` : Intention to acquire exclusive locks on finer granularity.
  - `SIX (Shared and Intention Exclusive)` : Acquires a shared lock and intention exclusive locks on finer granularity.
- **Shared Lock (S):** Allows the transaction to read but not modify the locked item.
- **Exclusive Lock (X):** Allows the transaction to read and modify the locked item.

Example:
Consider a database with a hierarchy of a database (DB), tables (T1, T2), and rows (R1, R2).

1. **Transaction T1 wants to read Table T1:**
   - Acquires `IS` lock on DB.
   - Acquires `S` lock on T1.
2. **Transaction T2 wants to update Row R1 in Table T1:**
   - Acquires `IX` lock on DB.
   - Acquires `IX` lock on T1.
   - Acquires `X` lock on R1.