DBMS QB ANSWERS PT II

(1) Explain ACID properties of transaction.

Ans:- A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation and **D**urability commonly known as ACID properties - in order to ensure accuracy, completeness, and data integrity.

**Atomicity** − A transaction must be treated as an atomic unit, (either all of its operations are executed or none) There must be no state in a database where a transaction is left partially completed.

**Example:** We have two accounts A and B, each containing Rs 1000/-. We now start a transaction to deposit Rs 100/- from account A to Account B.

**Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database.

**Example:** In the internal fund transfer i.e. from account A to account B, the total amount of account A and account B must be same as before the transaction successfully executed.

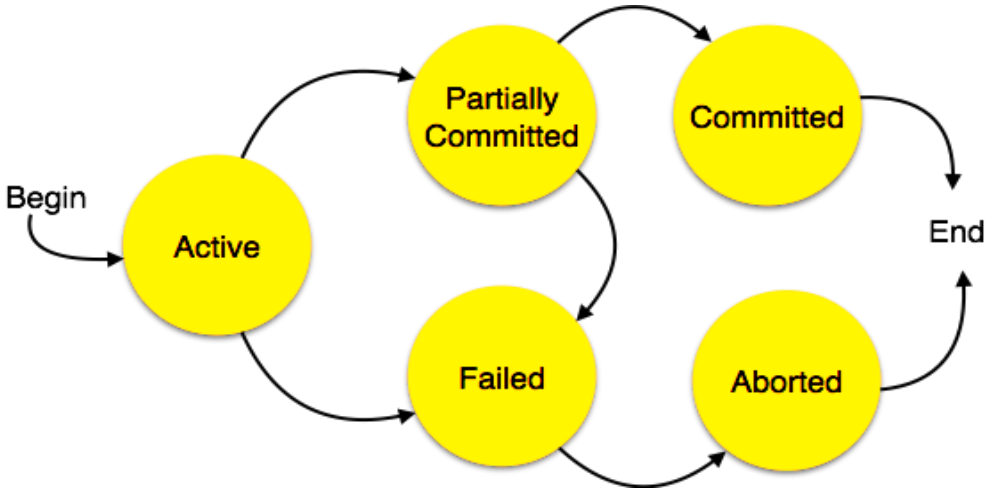
**Isolation** − The property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.

**Example**: For every pair of transactions Ti and Tj , it appears to Ti that either Tj, finished execution before Ti started, or Tj started execution after Ti finished.

**Durability − The database should be durable enough to hold all its latest updates even if the system fails or restarts.**

**Example:** Let us consider account A has balance of $300. Now $100 have been credited in account A then if power failure occurred after COMMIT then when the system becomes operable again then account A must contain $400.

(2) Draw and explain state diagram of transactions.




- **Active** − In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** − When a transaction executes its final operation, it is said to be in a partially committed state. The values generated during the execution are all stored in volatile storage.
- **Failed** − If the transaction fails for some reason. The temporary values are no longer required, and the transaction is set to **ROLLBACK**. It means that any change made to the database by this transaction up to the point of the failure must be undone. If the failed transaction has withdrawn Rs. 100/- from account A, then

the ROLLBACK operation should add Rs 100/- to account A.

- **Aborted** –If checks fail and the transaction has reached a failed state, the recovery manager rolls back all its write operations to bring the database back to its original state. Transactions in this state are called aborted. The database recovery module can either **re-start the transaction** (if due to hardware/software error) or **kill the transaction** (if due to logical error, bad input, or missing data).

(3) Define Transaction with an example.

Ans:-

➢A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

➢ Example:

**A's Account**

Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)

**B's Account**

Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)

(4) Explain the steps involved in query processing and optimization.

Ans:- Query optimization involves three steps:

**1. Query Tree Generation** − A query tree is a tree data structure representing a relational algebra expression. Tables are represented as leaf nodes, relational algebra operations as internal nodes, and the root represents the query as a whole. Execution proceeds by replacing nodes with result tables until the root is executed.

**2. Query Plan Generation** − After the query tree is generated, a query plan is made. It is an extended query tree that includes access paths for all operations. It also states how intermediate tables should be passed, how temporary tables should be used, and how operations should be pipelined/combined.

**3. Code Generation** − This is the final step. It produces the executable form of the query depending on the operating system. Once generated, the Execution Manager runs it and produces the results.

(5) Give the advantages of concurrency in DBMS.

Ans:-

**Minimization of Response Time**
Concurrency allows multiple queries to be processed simultaneously, reducing the time it takes for users to get results.

**Maximized System Throughput**
By executing several queries in parallel, the system can handle more requests in a given time frame, increasing overall efficiency.

**Reduced Resource Usage**
Efficient concurrent execution can lower the memory and storage requirements during query processing.

**Increased Parallelism**
This is a direct benefit of concurrency—operations can be distributed across processors or cores, speeding up execution.

(6) Explain Timestamp Based Protocols.

Ans:- With each transaction **Ti** in the system, we associate a unique fixed timestamp, denoted by **TS(Ti)**. This timestamp is assigned by the database system before the transaction Ti starts execution. If a transaction Ti has been assigned timestamp TS(Ti), and a new transaction Tj enters the system, then **TS(Ti) < TS(Tj)**.

Timestamps may be assigned using the **system clock** or a **logical counter**.

- To implement this scheme, we associate with each data item $Q$ two timestamp values:

  • **W-timestamp**($Q$) : denotes the largest timestamp of any transaction that executed write($Q$) successfully.

  • **R-timestamp**($Q$): denotes the largest timestamp of any transaction that executed read($Q$) successfully.

These values are updated on every read(Q) or write(Q).

**Timestamp-Ordering Protocol**

This protocol ensures that conflicting operations are executed in timestamp order:

1. **read(Q) by Ti**
    - If **TS(Ti) < W-timestamp(Q)** → read rejected, Ti rolled back.
    - Else → read executed and R-timestamp(Q) updated.
2. **write(Q) by Ti**
    - If **TS(Ti) < R-timestamp(Q)** or **TS(Ti) < W-timestamp(Q)** → write rejected, Ti rolled back.
    - Else → write executed and W-timestamp(Q) updated.

If a transaction is rolled back, the system assigns it a **new timestamp** and restarts it.

(7) Explain Validation based protocol.

Ans:- This protocol is also called as **Optimistic Concurrency Control Protocol** because it is based on the assumption that conflicts are rare. The transaction is executed without any checking, but at the end, validation is done.

There are three phases for this concurrency control protocol:

1. **Read phase:** A transaction can read values of committed data items from the database.

2. **Validation phase:** Checking is performed to ensure that serializability will not be violated if the transaction updates are applied to the database.
3. **Write phase:** If the validation phase is successful, the transaction updates are applied to the database; otherwise, the updates are discarded and the transaction is restarted

**Example**

- Suppose **T1** and **T2** are two transactions.
- **T1** completes its read and enters validation phase. **T2** is still in read phase.
- If validation shows that **T1's write does not conflict with T2's read/write**, then T1 is allowed to commit and write its updates to the database.
- But if validation fails (e.g., both T1 and T2 updated the same data item), then **T1 is rolled back** and restarted.

(8) Explain the concepts of LOCK in concurrency control.

Ans:- Locks are mechanisms used in a database to control concurrent access to data items.

- A **lock** is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.
- Two types of locks:
    1. **Shared (S) Lock** − If a transaction Ti has obtained a shared-mode lock on data item Q, then Ti can read Q but cannot write Q.

2. **Exclusive (X) Lock** − If a transaction Ti has obtained an exclusive-mode lock on data item Q, then Ti can both read and write Q.

Locks are requested and released by transactions. Whether a lock is granted depends on the **lock-compatibility matrix**.

**Example**

Consider two transactions **T1** and **T2** working on a bank account balance.

- **T1:** Deposit Rs. 500 into account balance.
- **T2:** Withdraw Rs. 1000 from account balance.

If both transactions run concurrently **without proper locks**, the following may happen:

- T1 reads balance = 10,000
- T2 reads balance = 10,000
- T1 adds 500 → new balance = 10,500
- T2 subtracts 1000 → new balance = 9000
- Final balance = 9000 (incorrect, because one update is lost).

With **locks**, if T1 obtains an **X-lock** on balance, T2 must wait until T1 releases the lock. Then, after T1 commits, T2 executes. The correct final balance will be 9500.

✅ This shows why locks are necessary to avoid problems like **lost updates**.

(9) Explain starvation of transaction. Also state steps to avoid starvation.

Ans:- In a transaction-management context, starvation (sometimes called indefinite postponement) is the situation in which a transaction waits for a resource (lock) for an indefinite amount of time.

**Starvation of a transaction occurs because of the following reasons −**

- If higher priority transactions are continuously arriving and requesting for the same resource that the lower priority transaction has requested.
- If waiting transactions are repeatedly rolled back due to deadlock and restarted, a particular transaction may never get executed.

**We can avoid starvation** of transactions by granting locks in the following manner:

**1.** There is no other transaction holding a lock on *Q* in a mode that conflicts with *M*.

**2.** There is no other transaction that is waiting for a lock on *Q* and that made its lock request before *Ti* .

(10) Describe two-phase locking protocol.

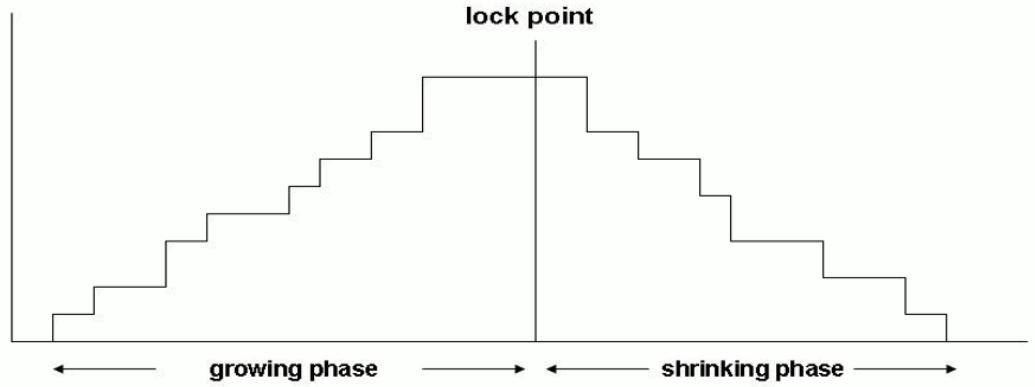
Ans:-

## THE TWO-PHASE LOCKING PROTOCOL

One protocol that ensures serializability is the **two-phase locking protocol**. This protocol requires that each transaction issue lock and unlock requests in two phases:

**1. Growing phase**. A transaction may obtain locks, but may not release any lock.

**2. Shrinking phase**. A transaction may release locks, but may not obtain any new locks.




Initially, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests.

(11)Define deadlock with the help of an example.

Ans:- In a multi-programming environment, **deadlock is a situation where two or more transactions are waiting indefinitely for one another to release locks.**

---

**Example**

- **Transaction T1** holds a lock on some rows in the **STUDENT** table and needs to update some rows in the **COURSE** table.

- **Transaction T2** holds a lock on some rows in the **COURSE** table and needs to update some rows in the **STUDENT** table.

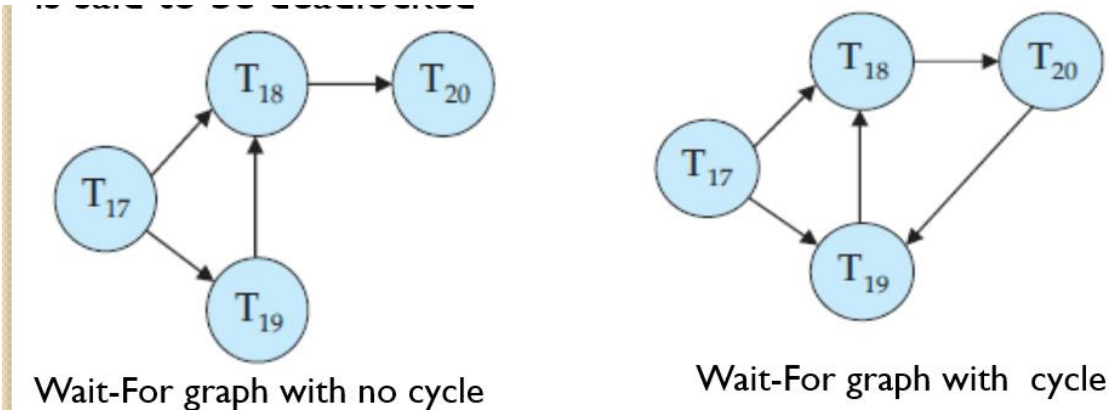
Now, T1 is waiting for T2 to release its lock, and T2 is waiting for T1 to release its lock. Neither ever releases the locks, so they both keep waiting forever.

(12) Explain wait-for graph for deadlock detection.

Ans:-

- Deadlocks can be described in terms of a directed graph called a **wait-for graph**.
- The nodes of the graph are the **transactions** {T1, T2, …, Tn}.
- There is a directed edge from Ti to Tj if transaction **Ti is waiting for Tj** to release a data item.

**Deadlock exists in the system if and only if the wait-for graph contains a cycle**




(13) Explain conflict serializibility with example.

Ans:-  A schedule is called **conflict serializable** if it can be transformed into a serial schedule by swapping **non-conflicting operations**.

 **Conflicting operations**: Two operations are said to be conflicting if all the following hold:

1. They belong to different transactions.
2. They operate on the same data item.
3. At least one of them is a write operation.

**Types of conflicts**

1. **read(Q) – read(Q):** No conflict, order doesn't matter.
2. **read(Q) – write(Q):** Order matters (value read may change).
3. **write(Q) – read(Q):** Order matters.
4. **write(Q) – write(Q):** Order matters because only the last write persists.

## Example:

**read–read (No Conflict)**

- **T1:** read(A) → gets 100
- **T2:** read(A) → gets 100

Both read the same value, order doesn't matter. ✅ **No conflict**

**2. read–write (Conflict)**

- **T1:** read(A) → gets 100

- **T2:**write(A=200)
  - 👉 If T1 before T2 → T1 reads 100
  - 👉 If T2 before T1 → T1 reads 200

  Different result depending on order → **Conflict**

**3. write–read (Conflict)**

- **T1:** write(A=150)
- **T2:**read(A)
  - 👉 If T1 before T2 → T2 reads 150
  - 👉 If T2 before T1 → T2 reads 100

  Order changes what T2 reads → **Conflict**

**4. write–write (Conflict)**

- **T1:** write(A=200)
- **T2:**write(A=300)
  - 👉 If T1 before T2 → final A = 300
  - 👉 If T2 before T1 → final A = 200

  Final value depends on order → **Conflict**

(14) Describe view serializibility with example.

- Ans:- Like **conflict serializability**, view serializability also ensures that a **concurrent schedule** is equivalent to a **serial schedule**.
- But here, instead of looking at operation conflicts, we check if the **transactions "view" the database in the same way**.

A schedule is **view serializable** if it is **view equivalent** to a serial schedule.

## ✅ Conditions for View Equivalence

Two schedules are view equivalent if:

1. **Initial Reads:** If a transaction reads a value of a data item in one schedule, it should read the same value in the other schedule (either the initial value or the value written by the same transaction).
2. **Writes:** If a transaction writes the final value of a data item in one schedule, the same transaction must write the final value in the other schedule.
3. **Read-From:** If a transaction reads a value written by another transaction in one schedule, it should do the same in the other schedule.

## 🕮 Example

Consider two transactions:

- **T1:** read(A); write(A)
- **T2:** read(A); write(A)

**Schedule S1 (Serial)**

T1: read(A)

T1: write(A)

T2: read(A)

T2: write(A)

**Schedule S2 (Concurrent)**

T1: read(A)

T2: read(A)

T1: write(A)

T2: write(A)

👉 In both S1 and S2:

- Both transactions read the same initial value of A.
- Final value of A is written by **T2**.
- Therefore, **S2 is view equivalent to S1 → view serializable**.

(15) Explain cascade and cascadeless schedule.

Ans:- **1. Cascading Schedule (Cascading Rollback)**

- In a **cascading schedule**, one transaction depends on the results of another uncommitted transaction.
- If the first transaction fails/rolls back, then the dependent transactions must also be rolled back.
- This chain reaction is called **cascading rollback**.

**Example (Cascading):**

T1: write(A)

T2: read(A) ← depends on T1's write

👉 If **T1 aborts**, then **T2** must also rollback, because it read an uncommitted value.

### 2. Cascadeless Schedule

- In a **cascadeless schedule**, a transaction is allowed to **read a data item only after the transaction that last wrote it has committed**.
- This avoids cascading rollbacks.
- Safer than cascading schedules.

**Example (Cascadeless):**

T1: write(A) (commit T1)

T2: read(A) ← reads only after T1 commits

👉 Even if **T1 fails before commit**, **T2** has not yet read its value, so no rollback is needed.

(16) Enlist the key features of NoSQL database.

Ans:-

**Dynamic schema:** NoSQL databases do not have a fixed schema and can adapt to changing data structures without migrations.

**Horizontal scalability:** Designed to scale out by adding more nodes to handle large amounts of data and traffic.

**Different data models:**

- Document-based (e.g., MongoDB, stores JSON/BSON documents)
- Key-value-based (e.g., Redis, stores key-value pairs)
- Column-based (e.g., Cassandra, organizes data in columns)

**Distributed and high availability:** Handle replication and node failures automatically across clusters.

**Flexibility:** Support multiple data types and dynamic structures.

**Performance:** Optimized for high performance with large volumes of reads and writes (big data, real-time apps).

(17) State advantages and disadvantages of using NoSQL databases.

Ans:-

### ✅ Advantages of NoSQL

- Scalable (horizontal scaling, sharding).
- Flexible (handles unstructured/semi-structured data).
- High availability (replication, fault tolerance).
- High performance (fast reads/writes, real-time apps).
- Cost-effective & good for agile development.

---

### ❌ Disadvantages of NoSQL

- Not fully ACID (eventual consistency).
- No standardization (many types).
- Limited support for complex queries.
- Less mature & harder to manage.
- Backup issues and large document size in some systems.

(18) Differentiate between SQL and NoSQL databases briefly.

Ans:-

| **SQL (Relational DB)** | **NoSQL (Non-Relational DB)** |
|---|---|
| SQL databases store data in fixed tables with rows and columns. | NoSQL databases store data in a flexible way like documents, key-value, or graphs. |
| They use a fixed schema, so data structure cannot change easily. | They use a fixed schema, so data structure cannot change easily. |

| SQL (Relational DB) | NoSQL (Non-Relational DB) |
|---|---|
| They follow ACID properties and give strong consistency. | They often provide eventual consistency, not always full ACID. |
| SQL databases scale vertically by upgrading the same server. | NoSQL databases scale horizontally by adding more servers. |
| They are best for structured data and complex queries like JOINs. | They are best for unstructured or big data with fast reads and writes. |
| Examples are MySQL, Oracle, and PostgreSQL. | Examples are MongoDB, Cassandra, and Redis. |

(19) List two common types of NoSQL databases.

Ans:-

📌 **Types of NoSQL Databases**

1. **Document Databases** – Store data in JSON-like documents. Example: **MongoDB**.
2. **Key-Value Databases** – Store data as simple key–value pairs. Example: **Redis**.
3. **Wide-Column Stores** – Store data in tables, rows, and dynamic columns. Example: **Cassandra**.
4. **Graph Databases** – Store data as nodes and edges to represent relationships. Example: **Neo4j**.

---

👉 Easy trick to remember: **D–K–W–G (Document, Key-value, Wide-column, Graph).**

(20) Give compatibility function.

Ans:-The compatibility function determines whether a transaction requesting a lock can be granted immediately, based on the lock mode already held by another transaction.

- **Formal Expression**

  If transaction **Ti** requests a lock of mode **A** on data item **Q**, and transaction **Tj** already holds a lock of mode **B** on **Q**,then:
  **Mode A is compatible with mode B** if **Ti** can be granted the lock immediately.

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

Lock – Compatibility Matrix

This matrix is used by the concurrency control manager to decide whether to allow or delay lock requests.

(21) State the significance of SQL in Cassandra.

Ans:-

📌 **Significance of SQL in Cassandra**

- **CQL offers a model similar to SQL**, making it easier for users familiar with relational databases to interact with Cassandra.
- Although Cassandra is a NoSQL database, **CQL provides SQL-like syntax** for defining tables, inserting data, and querying records.
- The data in Cassandra is stored in **tables containing rows and columns**, just like in SQL databases.
- This SQL-like approach helps bridge the gap between traditional relational database users and Cassandra's distributed architecture.

(22)Describe the basic steps involved in creating a new database in Cassandra.

Ans:-

**Steps to Create a New Database in Cassandra**

1. **Create Keyspace –**

```sql
CREATE KEYSPACE <keyspace_name>
WITH replication = {'class':'SimpleStrategy','replication_factor':1};
```

2. **Use Keyspace –**

```sql
USE <keyspace_name>;
```

3. **Alter Keyspace –**

```sql
ALTER KEYSPACE <keyspace_name> WITH <properties>;
```

4. **Drop Keyspace –**

```sql
DROP KEYSPACE <keyspace_name>;
```

(23)Describe the basic steps involved in creating a new database in MongoDB.

**Ans:-**

### 🌐 Steps to Create a New Database in MongoDB

1. **Start MongoDB Shell**
   - Open terminal/command prompt and run:

```nginx
mongo
```

2. **Create/Select Database**
   - Use the `use` command.
   - If the database does not exist, MongoDB will create it once you insert data.

```js
use mydb
```

3. **Create Collection (like a table)**
   - Collections are created automatically when you insert data.
   - Or explicitly:

```js
```

4. **Insert Documents (records)**
   - Insert data into the collection:

```js
db.students.insertOne({name:"Ravi", age:20})
```

5. **Verify Database**
   - Show all databases:

```js
show dbs
```

   - Show current database:

```js
db
```

(24) Describe the steps involved in setting up a MongoDB database, creating collections, and inserting documents into a collection.

Ans:-

### 🌐 Setting up a MongoDB Database

#### 1. Create/Select Database

```js
use mydb
```

- If `mydb` exists, it switches to it.
- If not, MongoDB creates it after first data insertion.

---

#### 2. Create Collection

- Either explicitly:

```js
db.createCollection("students")
```

- Or automatically (when inserting a document).

### 3. Insert Documents

- Insert a single document:

```js
db.students.insertOne({name: "Ravi", age: 20})
```

- Insert multiple documents:

```js
db.students.insertMany([{name: "Sneha", age: 21}, {name: "Amit", age: 22}])
```

---

### 4. Verify

- Show all databases:

```js
show dbs
```

- Show current database:

```js
db
```

- Show collections:

```js
show collections
```

- Display data:

```js
db.students.find()
```