USN | | | | | | | | | | |

Internal Assessment Test 3 – August 2024

Sub:	**Database Management Systems**					Sub Code:	BCS403	Branch:	ISE	
Date:	**04/06/2024**	Duration:	90 min's	Max Marks:	50	Sem/Sec:	IV A, B & C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a. What are ACID properties? Explain with an example. b. How do you test conflict serializability of a schedule S? Explain with the help of a graph.	5 5	CO5	L2
2	Why is concurrency control needed? Demonstrate with an example	10	CO5	L2
3	Discuss binary locks and unlock operations & shared/ exclusive operations	10	CO5	L2
4	Explain timestamp ordering algorithm protocol for concurrency control.	10	CO5	L2
5	a. What is NOSQL? Explain the CAP theorem. b. What is NOSQL Graph database? Explain Neo4j.	5 5	CO6	L2
6	What are document based NOSQL systems? basic operations CRUD in MongoDB.	10	CO6	L2

CI CCI HOD

USN | | | | | | | | | | |

Internal Assessment Test 3 – August 2024

Sub:	**Database Management Systems**					Sub Code:	BCS403	Branch:	ISE	
Date:	**04/06/2024**	Duration:	90 min's	Max Marks:	50	Sem/Sec:	IV A, B & C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a. What are ACID properties? Explain with an example. b. How do you test conflict serializability of a schedule S? Explain with the help of a graph.	5 5	CO5	L2
2	Why is concurrency control needed? Demonstrate with an example	10	CO5	L2
3	Discuss binary locks and unlock operations & shared/ exclusive operations	10	CO5	L2
4	Explain timestamp ordering algorithm protocol for concurrency control.	10	CO5	L2
5	a. What is NOSQL? Explain the CAP theorem. b. What is NOSQL Graph database? Explain Neo4j.	5 5	CO6	L2
6	What are document based NOSQL systems? basic operations CRUD in MongoDB.	10	CO6	L2

CI CCI HOD

USN | | | | | | | | | | |

Internal Assessment Test 3 – August 2024
Scheme of Evaluation and Solution

Sub:	**Database Management Systems**					Sub Code:	BCS403	Branch:	ISE	
Date:	**05/08/2024**	Duration:	90 min's	Max Marks:	50	Sem/Sec:	IV A, B & C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	**c. What are ACID properties? Explain with an example.**		CO5	L2
	Explanation of ACID properties (1M) **Atomicity** (1M) Ensures that all operations within a transaction are either fully completed or not executed at all. Example: In a banking system, if a transaction involves transferring money from Account A to Account B, either both the debit from Account A and the credit to Account B happen, or neither happens. **Consistency:** (1M) Ensures that a transaction takes the database from one valid state to another. Example: In the banking system, the sum of balances in Account A and Account B should remain unchanged after the transfer. **Isolation:** (1M) Ensures that concurrent execution of transactions leaves the database in the same state that would be obtained if the transactions were executed sequentially. Example: While transferring money from Account A to Account B, another transaction trying to access Account A or Account B will wait until the transfer is complete. **Durability:** (1M) Ensures that once a transaction is committed, it will remain committed even in the case of a system failure. Example: Once the money transfer is confirmed, the changes are permanent and will persist even if the system crashes immediately after.	5		
	d. **How do you test conflict serializability of a schedule S? Explain with the help of a graph.** Testing conflict serializability of a schedule involves determining if a schedule can be transformed into a serial schedule through a series of equivalent operations. Conflict serializability ensures that a schedule produces the same outcome as some serial schedule, where transactions are executed sequentially. **(1M)** **Steps to Test Conflict Serializability** **(2M)** **1. Construct the Precedence Graph:** o Nodes: Each transaction in the schedule becomes a node in the graph. o Edges: Draw directed edges between nodes based on conflicting operations. A conflict occurs if: ▪ Two operations are from different transactions. ▪ They access the same data item. ▪ At least one of the operations is a write. **2. Analyze the Graph:** o Cycles: Check if the graph contains any cycles. If there are no cycles, the schedule is conflict-serializable. If there is a cycle, the schedule is not conflict-serializable. **Example with a Graph Schedule S:** **(2M)** T1: R(A), W(A), R(B), W(B) T2: R(A), W(B) Conflicting Operations: • T1's W(A) conflicts with T2's R(A) (T1 → T2). • T1's W(B) conflicts with T2's W(B) (T2 → T1). **Steps to Test Conflict Serializability:** **1. Identify Conflicting Operations:** o W(A) in T1 before R(A) in T2. o W(B) in T2 before W(B) in T1. **2. Construct the Precedence Graph:** o Nodes: T1, T2. o Edges: ▪ Draw an edge from T1 to T2 due to W(A) in T1 before R(A) in T2. ▪ Draw an edge from T2 to T1 due to W(B) in T2 before W(B) in T1. The precedence graph will look like this:	5		

	T1 ↑ | ↓ T2 There is a directed edge from T1 to T2 and from T2 to T1, forming a cycle. **3. Analyze the Graph:** o Cycle Detection: The graph contains a cycle (T1 → T2 → T1).			
2	**Why is concurrency control needed? Demonstrate with an example** Concurrency control is necessary to ensure the consistency and isolation properties of transactions. **(1M)** Without it, the following issues can occur: • **Lost Updates:** When two transactions read the same data and then update it, one update can be lost. . **(1M)** • **Dirty Read:** Happens when a transaction reads data that has been written by another transaction that is not yet committed, leading to potential inconsistencies. **(1M)** • **Incorrect Summary:** Arises when a transaction calculates an aggregate summary of data while another transaction is updating some of that data, resulting in an inaccurate summary.. **(1M)** • **Unrepeatable Read:** Happens when a transaction reads the same data twice and gets different results each time due to another transaction modifying the data in between the reads.. **(1M)** Examples of all issues. **(5M)**	10	CO5	L2
3	**Discuss binary locks and unlock operations & shared/ exclusive operations** **Binary Locks:** **(5M)** 1. **Definition:** o A binary lock is a simple type of lock that can be in one of two states: locked (1) or unlocked (0). 2. **Operations:** o **Lock (Acquire):** This operation sets the lock on a data item. If the data item is already locked, the transaction requesting the lock must wait until it is released. o **Unlock (Release):** This operation releases the lock on a data item, allowing other transactions to acquire the lock. 3. **Behavior:** o When a transaction holds a binary lock on a data item, no other transaction can access that data item until the lock is released. o Binary locks provide a straightforward mechanism for ensuring mutual exclusion, where only one transaction can access a data item at a time. 4. **Example:** o Suppose Transaction T1 locks data item A using a binary lock. No other transaction can read or write to A until T1 releases the lock. **Shared/Exclusive (S/X) Locks** **Shared (S) Locks:** **(3M)** 1. **Definition:** o A shared lock allows multiple transactions to read a data item but prevents any transaction from writing to it. 2. **Operations:** o **Acquire Shared Lock:** A transaction can acquire a shared lock on a data item if no other transaction holds an exclusive lock on the same data item. o **Release Shared Lock:** The transaction releases the shared lock after it completes reading the data item. 3. **Behavior:** o Multiple transactions can hold shared locks on the same data item simultaneously. o Shared locks are used to enable concurrent read operations without interference. 4. **Example:** o Transactions T1 and T2 can both hold shared locks on data item A and read it concurrently. **Exclusive (X) Locks:** **(2M)** 1. **Definition:** o An exclusive lock allows a transaction to read and write a data item, but no other transaction can access the data item while the exclusive lock is held. 2. **Operations:**	10	CO5	L2

- **Acquire Exclusive Lock:** A transaction can acquire an exclusive lock on a data item if no other transaction holds either a shared or exclusive lock on the same data item.
- **Release Exclusive Lock:** The transaction releases the exclusive lock after it completes its operations on the data item.

3. **Behavior:**
   - Only one transaction can hold an exclusive lock on a data item at a time.
   - Exclusive locks provide mutual exclusion for both read and write operations.
4. **Example:**
   - If Transaction T1 holds an exclusive lock on data item A, no other transaction can read or write to A until T1 releases the lock.

Lock Compatibility Matrix

The compatibility matrix helps in determining whether a transaction can acquire a lock based on the locks held by other transactions:

Compatibility	**S Lock**	**X Lock**
S Lock	Yes	No
X Lock	No	No

- **S Lock:** If a data item has a shared lock, other transactions can also acquire a shared lock on it.
- **X Lock:** If a data item has an exclusive lock, no other transaction can acquire any type of lock on it.

4	**Explain timestamp ordering algorithm protocol for concurrency control.** The Timestamp Ordering (TO) protocol is used in database systems to ensure that transactions are executed in a serializable order based on their timestamps. This protocol assigns a unique timestamp to each transaction when it starts, and this timestamp determines the order in which the transactions must be executed. **(2M)** **Key Concepts** **(2M)** 1. **Timestamps:** o Each transaction TiT_iTi is given a unique timestamp TS(Ti)TS(T_i)TS(Ti) when it starts. o These timestamps are used to order the transactions. If TS(Ti)<TS(Tj)TS(T_i) < TS(T_j)TS(Ti)<TS(Tj), then TiT_iTi must appear to execute before TjT_jTj. 2. **Read and Write Timestamps:** o Each data item QQQ in the database has two timestamps: ▪ **WTS(Q):** The timestamp of the last transaction that wrote to QQQ. ▪ **RTS(Q):** The timestamp of the last transaction that read QQQ. **Protocol Rules** **(4M)** The TO protocol ensures serializability by enforcing the following rules when a transaction TiT_iTi issues a read or write operation on a data item QQQ: 1. **Read Operation:** o When TiT_iTi issues a read(Q), the protocol checks if TS(Ti)<WTS(Q)TS(T_i) < WTS(Q)TS(Ti)<WTS(Q): ▪ If TS(Ti)<WTS(Q)TS(T_i) < WTS(Q)TS(Ti)<WTS(Q), it means that TiT_iTi is trying to read a value that has been overwritten by a later transaction TjT_jTj (where TS(Tj)>TS(Ti)TS(T_j) > TS(T_i)TS(Tj)>TS(Ti)). Hence, TiT_iTi's read operation is rejected, and TiT_iTi is aborted. ▪ If TS(Ti)≥WTS(Q)TS(T_i) \geq WTS(Q)TS(Ti)≥WTS(Q), the read operation is allowed to proceed. Additionally, RTS(Q)RTS(Q)RTS(Q) is updated to max(RTS(Q),TS(Ti))max(RTS(Q), TS(T_i))max(RTS(Q),TS(Ti)). 2. **Write Operation:** o When TiT_iTi issues a write(Q), the protocol checks two conditions: ▪ If TS(Ti)<RTS(Q)TS(T_i) < RTS(Q)TS(Ti)<RTS(Q), it means that QQQ has already been read by a later transaction TjT_jTj (where TS(Tj)>TS(Ti)TS(T_j) > TS(T_i)TS(Tj)>TS(Ti)). Allowing TiT_iTi to write QQQ would violate the serializability. Hence, TiT_iTi's write operation is rejected, and TiT_iTi is aborted. ▪ If TS(Ti)<WTS(Q)TS(T_i) < WTS(Q)TS(Ti)<WTS(Q), it means that QQQ has already been written by a later transaction TjT_jTj (where TS(Tj)>TS(Ti)TS(T_j) > TS(T_i)TS(Tj)>TS(Ti)). Allowing TiT_iTi to write QQQ would also violate the serializability. Hence, TiT_iTi's write operation is rejected, and TiT_iTi is aborted.	10	CO5	L2

	▪ If both conditions are false, the write operation is allowed to proceed, and WTS(Q)WTS(Q)WTS(Q) is updated to TS(Ti)TS(T_i)TS(Ti). **Example** **(2M)** Consider three transactions T1T1T1, T2T2T2, and T3T3T3 with timestamps TS(T1)=1TS(T1) = 1TS(T1)=1, TS(T2)=2TS(T2) = 2TS(T2)=2, and TS(T3)=3TS(T3) = 3TS(T3)=3. 1. **Initial State:** o Data item QQQ has WTS(Q)=0WTS(Q) = 0WTS(Q)=0 and RTS(Q)=0RTS(Q) = 0RTS(Q)=0. 2. **Transaction Operations:** o T1T1T1 issues write(Q): ▪ Since TS(T1)=1TS(T1) = 1TS(T1)=1 and both conditions TS(T1)<RTS(Q)TS(T1) < RTS(Q)TS(T1)<RTS(Q) and TS(T1)<WTS(Q)TS(T1) < WTS(Q)TS(T1)<WTS(Q) are false, T1T1T1 writes QQQ, and WTS(Q)WTS(Q)WTS(Q) is updated to 1. o T2T2T2 issues read(Q): ▪ Since TS(T2)=2TS(T2) = 2TS(T2)=2 and TS(T2)≥WTS(Q)TS(T2) \geq WTS(Q)TS(T2)≥WTS(Q), T2T2T2 reads QQQ, and RTS(Q)RTS(Q)RTS(Q) is updated to 2. o T3T3T3 issues write(Q): ▪ Since TS(T3)=3TS(T3) = 3TS(T3)=3 and both conditions TS(T3)<RTS(Q)TS(T3) < RTS(Q)TS(T3)<RTS(Q) and TS(T3)<WTS(Q)TS(T3) < WTS(Q)TS(T3)<WTS(Q) are false, T3T3T3 writes QQQ, and WTS(Q)WTS(Q)WTS(Q) is updated to 3. **Advantages** • **Ensures Serializability:** The TO protocol ensures that the transactions are executed in a serializable order based on their timestamps. • **Avoids Deadlocks:** Since transactions are ordered by their timestamps, there are no circular wait conditions, thus avoiding deadlocks. **Disadvantages** • **Aborts:** The protocol can lead to a high number of transaction aborts, especially in systems with high contention. • **Overhead:** Maintaining and checking timestamps can introduce overhead in the transaction processing system.			
5	**c. What is NOSQL? Explain the CAP theorem.** NoSQL refers to a broad class of database management systems that differ from traditional relational database management systems (RDBMS). These databases are designed to handle large volumes of data, provide flexible data models, and enable horizontal scaling. **Key Characteristics of NoSQL:** **(1M)** 1. Schema-less Data Model: o Unlike relational databases, NoSQL databases do not require a fixed schema, allowing for more flexibility in how data is stored and managed. 2. Horizontal Scalability: o NoSQL databases are designed to scale out by adding more servers, making them suitable for handling massive amounts of data and high-traffic applications. 3. Variety of Data Models: o NoSQL databases support various data models, including document, key-value, column-family, and graph models, each optimized for different types of applications. 4. High Performance: o These databases are optimized for high read/write performance, often at the cost of strict consistency. Types of NoSQL Databases: 1. Document Stores: o Example: MongoDB, CouchDB o Store data in JSON-like documents. 2. Key-Value Stores: o Example: Redis, DynamoDB o Store data as key-value pairs. 3. Column-Family Stores: o Example: Cassandra, HBase o Store data in columns rather than rows. 4. Graph Databases: o Example: Neo4j, OrientDB o Store data in graph structures with nodes and edges.	5	CO6	L2

CAP Theorem **(3M)**

CAP Theorem (Consistency, Availability, Partition Tolerance):

The CAP theorem, also known as Brewer's theorem, states that a distributed database system can only achieve at most two of the following three guarantees simultaneously:

1. Consistency (C):
   - Every read receives the most recent write or an error.
   - Ensures that all nodes see the same data at the same time.
2. Availability (A):
   - Every request receives a response, without guarantee that it contains the most recent write.
   - Ensures that the database system is always operational and responds to requests.
3. Partition Tolerance (P):
   - The system continues to operate despite network partitions or communication breakdowns.
   - Ensures that the system can handle failures or partitions in the network that prevent some nodes from communicating with others.

CAP Theorem Implications: **(1M)**

- In a distributed system, when a network partition occurs, the system must choose between consistency and availability.
- CA (Consistency and Availability): These systems ensure consistency and availability but cannot tolerate partitions. They are generally centralized systems rather than distributed.
- CP (Consistency and Partition Tolerance): These systems ensure consistency and can tolerate partitions but may sacrifice availability. Example: HBase.
- AP (Availability and Partition Tolerance): These systems ensure availability and can tolerate partitions but may sacrifice consistency. Example: Cassandra, DynamoDB.

d. What is NOSQL Graph database? Explain Neo4j.

NoSQL Graph Database: **(1M)**

A NoSQL graph database is a type of database designed to handle data structured as a graph, with nodes (entities) and edges (relationships). Graph databases are optimized for traversing and querying these relationships, making them particularly suited for applications with interconnected data, such as social networks, recommendation systems, and network analysis.

Key Characteristics of Graph Databases: **(2M)**

Nodes and Edges:

Nodes represent entities (e.g., people, products) and edges represent relationships between these entities (e.g., friendships, purchases).

Properties:

Both nodes and edges can have properties (key-value pairs) that store relevant information.

Schema-less:

Similar to other NoSQL databases, graph databases are typically schema-less, allowing for flexible and dynamic data models.

Efficient Relationship Traversal:

Graph databases are optimized for traversing relationships, which makes complex queries about connections and paths very efficient.

Example of a NoSQL Graph Database: Neo4j

Neo4j:

Neo4j is one of the most popular and widely used graph databases. It is an ACID-compliant transactional database with native graph storage and processing. 5

Key Features of Neo4j: **(2M)**

Native Graph Storage:

Neo4j uses a native graph storage format, meaning that it stores data as graphs directly on disk, which optimizes for graph-related operations.

Cypher Query Language:

Neo4j uses Cypher, a powerful declarative query language specifically designed for querying and updating graph data.

ACID Compliance:

Neo4j ensures transactional integrity with ACID properties (Atomicity, Consistency, Isolation, Durability).

High Performance:

Neo4j is optimized for high-performance graph traversal and query execution.

Scalability:

Neo4j can scale horizontally by sharding the graph across multiple nodes and vertically by increasing the resources of the existing nodes.

6	**What are document based NOSQL systems? basic operations CRUD in MongoDB.**
	Document-Based NoSQL Systems: (5M)
	Document-based NoSQL systems store data in documents, typically using formats like JSON, BSON (Binary JSON), or XML. Each document is a self-contained unit of data that encapsulates fields and values, allowing for a flexible and hierarchical data structure. Unlike traditional relational databases, document-based systems do not require a fixed schema, making them highly adaptable to changing data models.
	Key Characteristics of Document-Based NoSQL Systems:
	1. Schema Flexibility:
	o Documents can have varying structures, allowing for different fields and data types within the same collection.
	2. Hierarchical Data Representation:
	o Documents can represent complex nested data structures, which can directly map to objects in application code.
	3. Scalability:
	o These systems are designed to scale horizontally by distributing data across multiple servers.
	4. Ease of Use:
	o Documents are easy to read and understand, making it simpler for developers to interact with the database.
	Popular Document-Based NoSQL Databases:
	• MongoDB
	• CouchDB
	• Amazon DocumentDB
	• ArangoDB
	MongoDB:
	MongoDB is one of the most widely used document-based NoSQL databases. It stores data in BSON format and provides powerful querying capabilities, indexing, and aggregation.
	Basic CRUD Operations in MongoDB
	CRUD stands for Create, Read, Update, and Delete. **(1M)**
	These are the four basic operations that can be performed on data in a database.
	1. **Create (Insert): (1M)**
	The insertOne and insertMany methods are used to add documents to a collection.
	Example:
	# Insert a single document
	db.collection.insertOne({ "name": "Alice", "age": 25, "city": "New York" })
	
	# Insert multiple documents
	db.collection.insertMany([
	{ "name": "Bob", "age": 30, "city": "Chicago" },
	{ "name": "Charlie", "age": 35, "city": "San Francisco" }])
	2. Read (Query): (1M)
	The find method is used to retrieve documents from a collection. It can take a query object to filter results.
	Example:
	# Find all documents
	db.collection.find()
	
	# Find documents with a specific condition
	db.collection.find({ "age": { "$gt": 30 } })
	
	# Find a single document
	db.collection.findOne({ "name": "Alice" })
	3. **Update: (1M)**
	The updateOne, updateMany, and replaceOne methods are used to modify existing documents.
	Example:
	# Update a single document
	db.collection.updateOne({ "name": "Alice" }, { "$set": { "age": 26 } })
	
	# Update multiple documents
	db.collection.updateMany({ "city": "Chicago" }, { "$set": { "city": "Boston" } })
	
	# Replace a document
	db.collection.replaceOne({ "name": "Charlie" }, { "name": "Charlie", "age": 36, "city": "Los Angeles" })
		10	CO6	L2

	4. Delete: **(1M)**
	The deleteOne and deleteMany methods are used to remove documents from a collection.
	Example:
	# Delete a single document
	db.collection.deleteOne({ "name": "Alice" })
	
	# Delete multiple documents
	db.collection.deleteMany({ "city": "Boston" })			

CI CCI HOD