VIT Vellore Institute of Technology (Deemed to be University under section 3 of the UGC Act, 1956)	**Final Assessment Test – Fall 2024-25 Semester - Nov 2024**	
	Course code : BCSE302L	Slot : A2 + TA2
	Course Title : Database Systems	Time: **Three Hours**
	Class Number(s) : VL2024250101539	Max. Marks: **100**
	Faculty Name : Dr. Shashank Mouli Satapathy	School: SCOPE
	Emp Id : 15257	Mobile No.: 8290979509

KEEPING MOBILE PHONE/ANY ELECTRONIC GADGETS, EVEN IN 'OFF' POSITION IS TREATED AS EXAM MALPRACTICE

General Instructions if any:

1. **Non Programmable calculator is permitted : NO**
2. **Reference tables are permitted : NO**

Answer ALL Questions

(10 X 10 = 100 Marks)

1. (Total Marks-10)

i) (4 Marks)

Two Definitions – 2 marks

Two Examples – 2 Marks

Database Schema:

- A database schema defines the structure of a database, including the tables, columns, data types, relationships, and constraints. It's essentially the blueprint of the database.
- **Example:** A schema for a customer database might include tables for customers, orders, and products, with columns for customer ID, name, address, order date, product ID, and quantity.

Database State:

- The database state refers to the actual data stored in the database at a particular point in time. It's the content of the database.
- **Example:** A database state for the customer database might include specific values for customer IDs, names, addresses, order dates, product IDs, and quantities.

ii) (6 Marks)

Difference with example – 3 marks

Challenges – 1.5 marks

Benefits – 1.5 marks

Logical vs. Physical Data Independence:

- **Logical Data Independence:** The ability to change the conceptual schema without affecting the external schema. This means that users can continue to access and manipulate data in the same way, even if the underlying structure of the database changes.
- **Physical Data Independence:** The ability to change the internal schema without affecting the conceptual schema. This means that the physical implementation of the database can be modified without impacting the way users interact with the data.

Example:

Consider a database for a university.

- **External Schema:** A student might only be able to view their personal information and course grades.
- **Conceptual Schema:** Defines entities like students, courses, and professors, with relationships between them.
- **Internal Schema:** Specifies how the data is stored in the database, including tables, indexes, and storage formats.

Challenges:

- **Complexity:** Implementing data independence can be complex, especially for large and complex databases.

- **Performance Overhead:** Data independence can sometimes introduce performance overhead, as mappings and translations need to be performed.
- **Schema Evolution:** Managing schema changes while maintaining data independence can be challenging, especially in evolving systems.

Benefits:

- **Flexibility:** Data independence allows for easier modifications to the database structure without affecting applications.
- **Data Portability:** Data can be migrated between different database systems more easily.
- **Data Security:** Data independence can help protect data from unauthorized access or modification.

2. (1st Option, Total Marks-10)

Identification of All entities and attributes with types – 3 marks
Represent Primary key and foreign keys – 2 marks
Establish appropriate relationship between entities – 2 marks
Cardinalities, Dependencies, Inheritance representation – 3 Marks

Entities & Attributes:

Product: ID (PK), name, description, price, quantity_in_stock, category_id
Category: ID (PK), name
Customer: ID (PK), name, email, address, shipping_information
Order: ID (PK), customer_id (FK), order_date, shipping_address, total_amount
OrderItem: ID (PK), order_id (FK), product_id (FK), quantity, price
Payment: ID (PK), order_id (FK), payment_method, transaction_id, payment_status
Review: ID (PK), product_id (FK), customer_id (FK), rating, comment

Relationships:

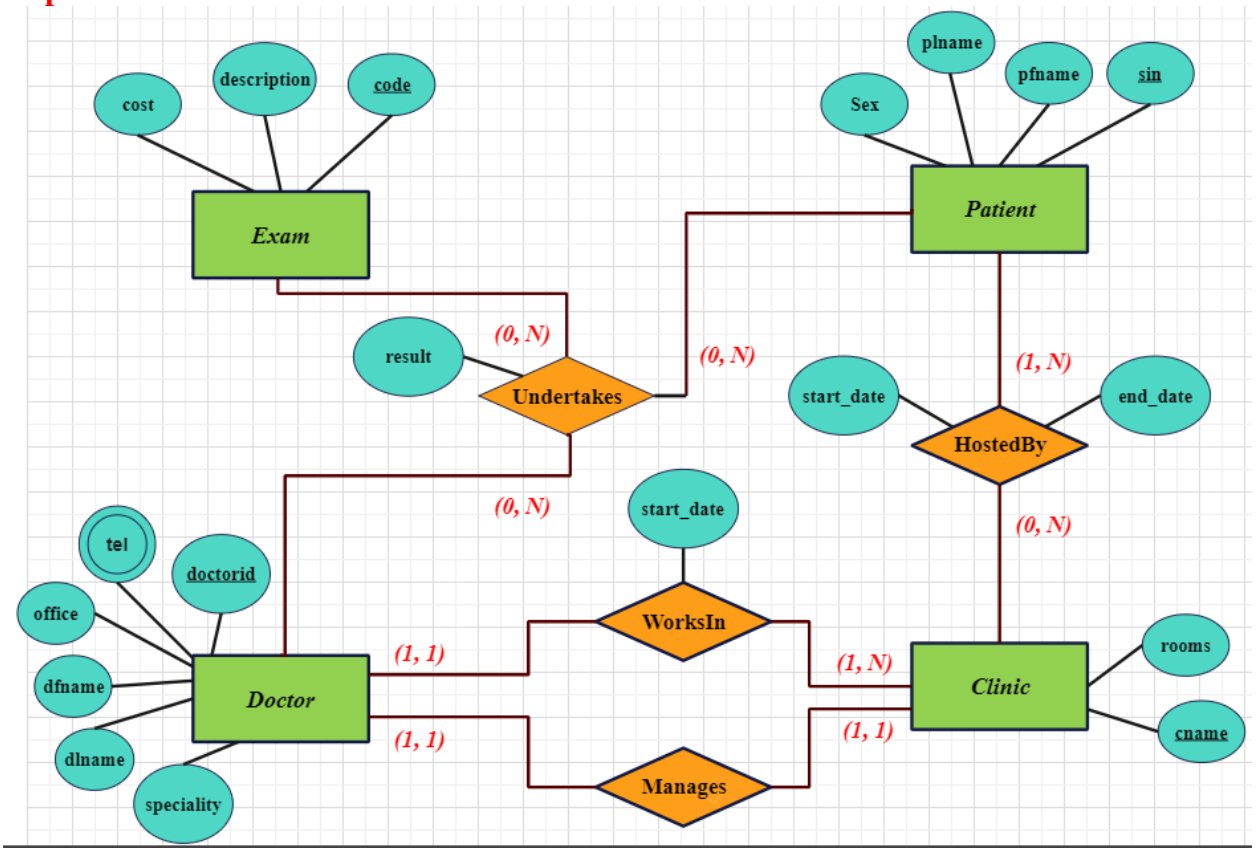
Product-to-Category: Many-to-one (a product belongs to one category, but a category can have many products)
Customer-to-Order: One-to-many (a customer can place many orders, but an order belongs to one customer)
Order-to-OrderItem: One-to-many (an order can have many items, but an item belongs to one order)
Product-to-Review: Many-to-many (a product can have many reviews, and a customer can leave reviews for multiple products)
Order-to-Payment: One-to-one (an order has one associated payment)

Strong Entities: Product, Category, Customer, Order, Payment, Review
Weak Entity: OrderItem (dependent on Order)

Inheritance: Product Could be further categorized into different types (e.g., books, electronics, clothing) using inheritance.

Full Dependency: The product_id attribute in the OrderItem entity is fully dependent on the order_id attribute.
Partial Dependency: The category_id attribute in the Product entity is partially dependent on the product_id attribute.

(2nd Option, Total Marks-10)
Identification of relations with attributes – 5 marks
Mapping of keys (primary and foreign) – 3 marks
Step wise solution – 2 marks



Doctor (<u>doctorId</u>, dfname, dlname, office, specialty, **worksin, manages**, start_date)
Tel(<u>doctorId</u>, tel)
Patient(<u>sin</u>, pfname, plname, sex)
Exam(<u>code</u>, description, cost)
Clinic(<u>cname</u>, rooms, **doctorId**)
Undertakes(**sin, code, doctorId**, result)
HostedBy(**sin, cname**, start_date, end_date)

<u>Underline</u> indicates a key
Bold red indicates a foreign key

3. (Total Marks-10)
Identification of key – 2 marks
Identification of Prime and Non-Prime attributes – 1 mark
Identification of present normal form with justification – 2 marks
Violation identification and decomposition to 2NF, 3NF and BCNF – 5 Marks
i) Identifying the Key:
I and S must be in any candidate key since they do not appear on the right of any f.d. The question is whether they form a complete candidate key. And yes, IS → ISDBOQ. Hence, the only candidate key is IS.

ii) List the prime and non-prime attributes
Prime = {I, S}
Non-prime = {B, O, Q, D}

iii) Identifying the present normal form:
The relation is in 1 NF

iv) Decomposing into 2NF:
R1 = {I, S, Q}
R2 = {I, B, O}
R3 = {S, D}

Decomposing into 3NF:
R1 = {I, S, Q}
R21 = {I, B}
R3 = {S, D}
R22 = {B, O}

Decomposing into BCNF:
No Further Decomposition as the Relation is already in BCNF.

4. (Total Marks-10)

i) (4 marks)

Given,
F1 = { A → B, C → DE, B → EC }
F2 = { A → B, A → C, B → D, C → E }

Check if F1 covers F2
{A+} = {A, B, E, C} wrt F1, A → B, A → C can be inferred from F1
{B+} = {B, E, C, D} wrt F1, B → D can be inferred from F1
{C+} = {C, D, E} wrt F1, C → E can be inferred from F1

Hence F1 Covers F2

Check if F2 covers F1
{A+} = {A, B, C, D, E} wrt to F2, A → B can be inferred from F2
{C+} = {C, E} wrt to F2, C → DE can't be derived from F2
{B+} = {B, D} wrt F2, B → EC can't be derived from F2

But F2 doesn't cover F1

Hence F1 and F2 are not equivalent

ii) (6 marks)

a) **Candidate key(s): BD.**
The decomposition into BC and AD is unsatisfactory because it is lossy (the join of BC and AD is the Cartesian product which could be much bigger than ABCD).

b) **Candidate key(s): A, C**
Since A and C are both candidate keys for R, it is already in BCNF. So from a normalization standpoint it makes no sense to decompose R further.

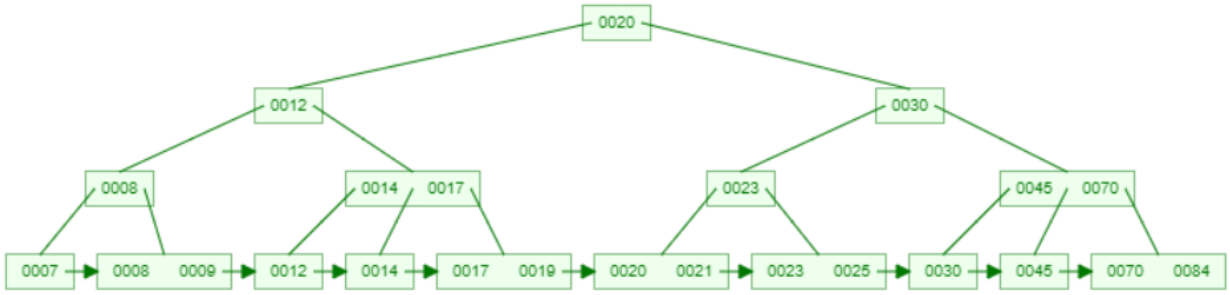
c) **Candidate key(s): A**
The projection of the dependencies on AB are: A → B and those on ACD are: A → C and C → D (rest follow from these). The scheme ACD is not even in 3NF, since C is not a superkey, and D is not part of a key. This is a lossless-join decomposition (since A is a key), but not dependency preserving, since B → C is not preserved.

5. (1st Option, Total Marks-10)

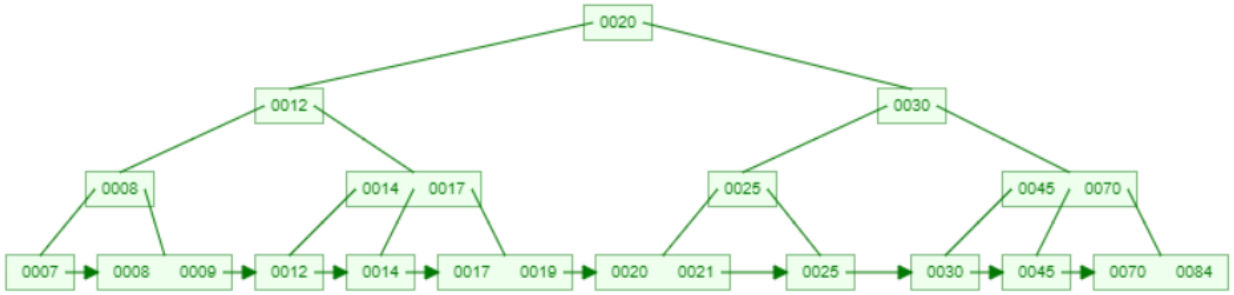
B+ tree Creation – 5 marks

Each Deletion – 1 mark, Total – 5 marks

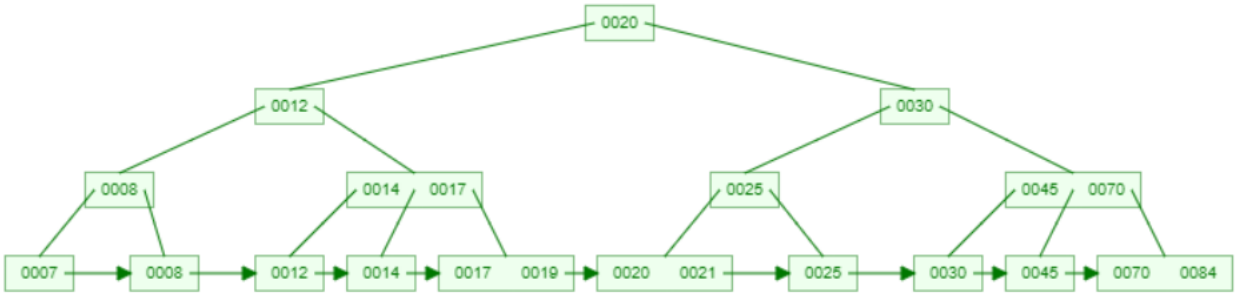
Create B+ tree of order 3



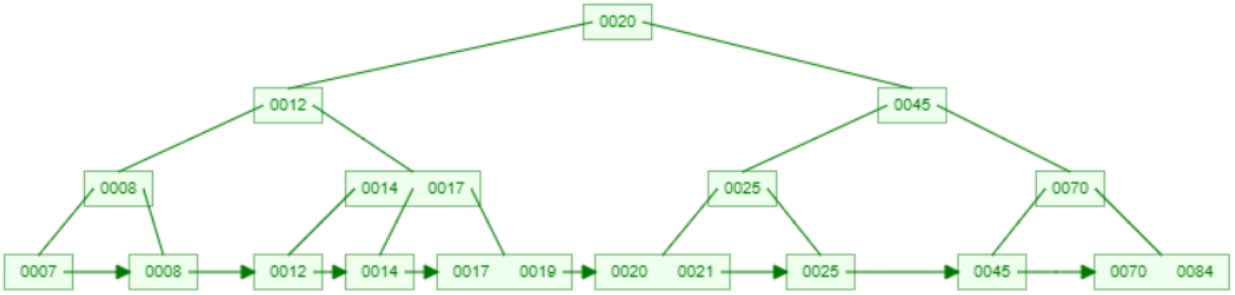
After Delete 23



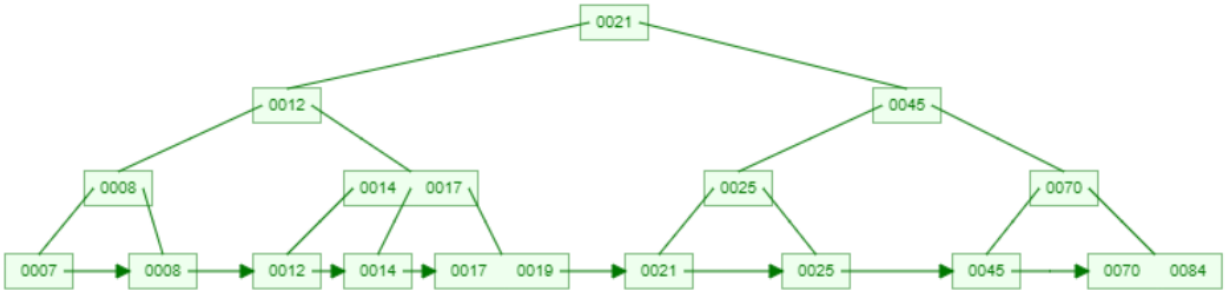
After Delete 9



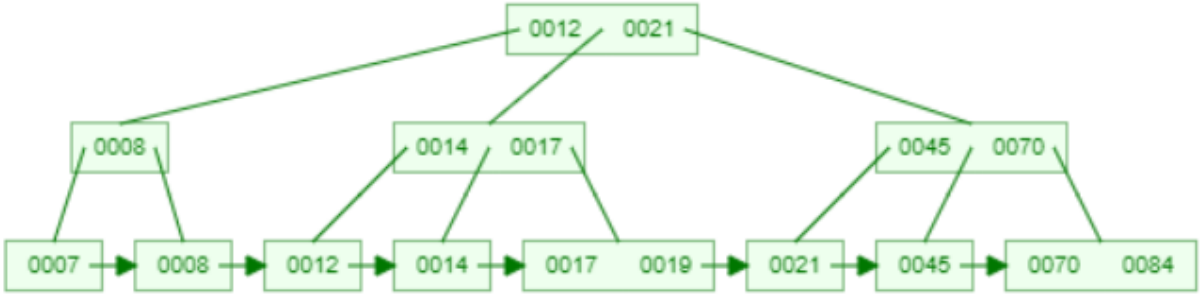
After Delete 30



After Delete 20



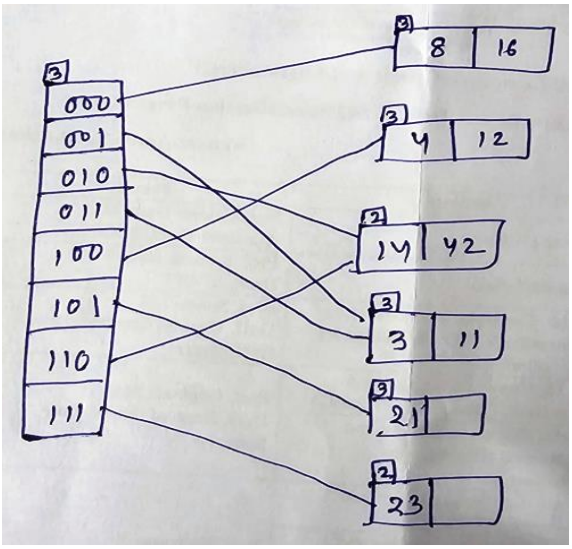
After Delete 25



5. (2nd Option, Total Marks-10)

i) (5 marks)
Extendable hash structure creation – 4 marks
Answer each question- 0.5 marks Total – 1 marks

a) 3
b) 2



ii) (5 marks)
Each answer – 1 mark, Total – 5 marks

a) $\pi_{first\_name,last\_name}$ (author ∗ author pub)
b) $\pi_{first\_name,last\_name}((\pi_{author\_id}(author) - \pi_{editor}(book)) * author)$
c) Which authors authored a publication that was published in July?
d) How many authors are not book editors?
e) 11

6. (Total Marks-10)

i) (6 marks)
Each question- 2 marks
Identification – 1 mark, Justification & example – 1 mark

a)
S = r1(A) r2(A) w1(A) w2(A)
P(S) = T2 → T1 → T2
The schedule is not conflict-serializable, since there is a cycle in the precedence graph.
To see that the schedule is also not serializable, consider the following example:
A = 10, T1: A:=A*10, T2: A:=A*10
The above interleaved schedule results in A = 100, but both serial schedules result in

A = 1000.

b)
S = r1(A) r2(B) w3(A) r2(A) r1(B)
P(S) = T1 → T3 → T2
The schedule is conflict-serializable, since its precedence graph does not contain any cycle.
The only conflict-equivalent serial schedule is T1, T3, T2.

The schedule is also serializable, since conflict-serializability implies serializability.

c)
S = r1(A) w2(A) w1(A) r3(A)
P(S) = T1 → T2 → T1 → T3
The schedule is not conflict-serializable, because its precedence graph is cyclic.
To see that the schedule is also not serializable, consider the following example:
A = 10, T1: A:= A * 10, T2: A:= 1000, T3: PRINT(A)
S results in A = 100 and PRINT 100. All possible serial schedules have a different net effect as follows:
T1, T2, T3: A = 1000, PRINT 1000
T1, T3, T2: A = 1000, PRINT 100
T2, T1, T3: A = 10000, PRINT 10000
T2, T3, T1: A = 10000, PRINT 1000
T3, T1, T2: A = 1000, PRINT 10
T3, T2, T1: A = 10000, PRINT 10.

ii) (4 marks)
For each scenario (identify right property and validation strategy) – 1 mark
a) Consistency
Scenario: A bank account has an initial balance of $100. A transaction transfers $50 to another account. The database must ensure that the total balance of all accounts remains consistent before and after the transaction.
Validation: Verify that the sum of all account balances remains unchanged after the transaction.

b) Durability
Scenario: The system experiences a power outage during a transaction. The transaction should be recoverable upon system restart.
Validation: Implement mechanisms like transaction logging and checkpointing to ensure that committed transactions are persisted to durable storage and can be recovered in case of failures.

c) Isolation
Scenario: Multiple customers are trying to purchase the last available item of a product. The system should ensure that only one customer is able to successfully purchase the item, preventing race conditions.
Validation: Use techniques like locking or timestamp-based concurrency control to prevent conflicts between concurrent transactions.

d) Atomicity
Scenario: A customer places an order for multiple items. The transaction should either be fully committed (all items are added to the order and payment is processed) or fully rolled back (no changes are made to the database).
Validation: Verify that the order status is either "completed" or "failed," with no partial updates.

7. (Total Marks-10)
Definition - 3 marks
For each schedule (Identification & Justification) – 1 mark, Total marks – 7 marks

• A schedule is conflict serializable if its precedence graph is acyclice.
• A schedule is recoverable if Tj reads an item written by Ti then Ti commits before Tj commits.
• A schedule is cascadeless if Tj reads an item written by Ti then Ti commits before Tj reads that item.
• A schedule is strict if a value written by a transaction T is not read or overwritten by other transactions until T either aborts or commits.

	C-Serializable	Recoverable	Cascadeless	Strict
a)	-	Y	Y	-
b)	-	Y	Y	-
c)	-	Y	Y	Y
d)	-	-	-	-
e)	Y	Y	-	-
f)	Y	Y	Y	Y

8. (Total Marks-10)

Analyze the trade-offs between binary locks, Exclusive lock, wait-die and wound-die - 5 marks
Scenario discussion – 2 marks
Explain the deadlock prevention process – 3 marks
Locking Mechanisms:

Binary Locks:
Concurrency: Low concurrency, as only one process can hold the lock at a time.

Performance: Can be efficient for simple operations, but can lead to contention and reduced performance in heavily contended scenarios.

Overhead: Low overhead compared to other locking mechanisms.

Exclusive/Shared Locks:
Concurrency: Higher concurrency than binary locks, as multiple processes can hold shared locks on the same data.

Performance: Can improve performance compared to binary locks, but still can introduce contention.

Overhead: Moderate overhead compared to binary locks.
Deadlock Prevention Protocols:

Wait-Die:
Concurrency: Can improve concurrency compared to a strict timeout-based approach.

Performance: May introduce additional overhead for checking timestamps and making decisions.

Overhead: Moderate overhead.

Wound-Wait:
Concurrency: Can improve concurrency compared to wait-die, as older transactions can preempt younger ones.

Performance: May introduce additional overhead for checking timestamps and making decisions.

Overhead: Moderate overhead.

Trade-offs:

Concurrency: Exclusive/shared locks and deadlock prevention protocols generally offer higher concurrency than binary locks.

Performance: The choice of locking mechanism and deadlock prevention protocol depends on the specific workload and performance requirements. In some cases, binary locks might be sufficient, while in others, more sophisticated mechanisms may be necessary.

Overhead: Deadlock prevention protocols introduce additional overhead for checking timestamps and making decisions.

Scenario:

A distributed database system is used to manage airline reservations. Two transactions are trying to book the last available seat on a flight:

Transaction A: Acquires a lock on the flight record.
Transaction B: Acquires a lock on the same flight record.
A deadlock occurs because neither transaction can proceed without releasing its lock, creating a circular dependency.

Deadlock Prevention:

Wait-Die: If a transaction requests a lock that is held by another transaction with a lower timestamp, it must wait. If a transaction requests a lock held by another transaction with a higher timestamp, it dies and is rolled back.

Wound-Wait: If a transaction requests a lock that is held by another transaction with a lower timestamp, it waits. If a transaction requests a lock held by another transaction with a higher timestamp and the younger transaction is waiting, the older transaction is rolled back (wounded).

9. (Total Marks-10)

ii) (5 marks)
Explanation – 3 marks
Example – 2 marks

Two-phase locking (2PL) is a concurrency control mechanism that ensures serializability by requiring transactions to acquire all their locks before releasing any. While effective for many database operations, it can be inefficient for indexes due to the following reasons:

High Contention: Indexes are frequently accessed for search operations, leading to high contention for locks. 2PL can result in long wait times and reduced performance.

Fine-Grained Locking: Indexes often require fine-grained locking to allow concurrent access to different parts of the index. 2PL can be cumbersome to implement for fine-grained locking.

Deadlock Potential: The high contention for locks in indexes can increase the likelihood of deadlocks, which can further degrade performance.

Example:

Consider a B+-tree index with multiple levels. If a transaction needs to insert a new key into the index, it must acquire locks on multiple nodes, including the leaf node, intermediate nodes, and

possibly the root node. Using 2PL, other transactions would have to wait for these locks to be released, potentially causing significant delays.

ii) (5 marks)
Adv and disadv. (atleast 2) of strict 2PL over others – 3 marks
Justification of why strict 2PL disallows the given schedule– 2 marks
Strict Two-Phase Locking (Strict 2PL)

Advantages:

Simplicity: Strict 2PL is relatively simple to implement and understand.
Guarantees Serializability: Strict 2PL ensures that the execution of concurrent transactions is equivalent to a serial schedule, preventing anomalies like lost updates, dirty reads, and cascading aborts.
Deadlock Freedom: Strict 2PL can help prevent deadlocks by ensuring that transactions acquire all their locks before releasing any.

Disadvantages:

Concurrency: Strict 2PL can be less concurrent than other locking mechanisms, as transactions may need to wait for locks that are held by other transactions.
Overhead: Strict 2PL can introduce overhead due to the need to acquire and release locks for each transaction.

Strict 2PL has two two rules:

i. If a transaction T wants to read (respectively, modify) an object, it first requests a shared (respectively, exclusive) lock on the object.

ii. All locks held by a transaction are released when the transaction is completed.

With strict 2PL, the above schedule is not possible. Indeed, T1 first acquires shared locks on X and Y . When T2 runs, it also acquires a shared lock on X. When it tries to acquire an exclusive lock before writing X, however, it blocks, waiting for T1 to release its lock, which will happen only when T1 commits. The above schedule is thus impossible with strict 2PL. In fact, the schedule leads to a deadlock: when T1 tries to write X, it also blocks waiting for T2 to release its shared lock. Now, both transactions are waiting for each other. We have a deadlock. When the deadlock is detected, the DBMS will abort one of the transactions, allowing the other one to commit and release its locks.

10. (Total Marks-10)
Identification of suitable NoSQL type – 2 marks
Justification (atleast 2 points) with scenario as example – 4 marks
Why other No SQL types are not suitable with justification – 4 Marks
Document databases (e.g., MongoDB, Couchbase) would be the most suitable NoSQL database type for a large-scale e-commerce application due to the following reasons:
1. Flexible Data Model:

- **Hierarchical Structure:** Document databases store data in a flexible, hierarchical structure, making it easy to represent complex relationships and nested data. This is ideal for e-commerce applications where products, customer information, and order details can have varying structures.

- **Schema-on-Read:** Document databases offer a schema-on-read approach, allowing for dynamic changes to the data structure without requiring a predefined schema. This flexibility is crucial for handling evolving business requirements and new product categories.

Example: A product in an e-commerce application might have nested fields for attributes like name, description, price, reviews, and images. Document databases can easily accommodate this hierarchical structure, while other NoSQL databases might require more complex modeling.

2. High Scalability:

- **Horizontal Scaling:** Document databases can scale horizontally by adding more servers to the cluster, ensuring high availability and low latency even under heavy loads.
- **Sharding:** Data can be partitioned across multiple servers (shards) to improve performance and scalability.

Example: An e-commerce application handling millions of users and products can distribute the data across multiple servers using sharding techniques to ensure fast response times and avoid bottlenecks.

3. Rich Query Capabilities:

- Document databases support flexible query mechanisms, allowing for efficient retrieval and analysis of data.
- This includes full-text search, geospatial queries, and aggregation functions.

Example: A customer can search for products based on keywords, price range, or location using full-text search and geospatial queries.

4. Strong Consistency Options:

- While some NoSQL databases prioritize eventual consistency, document databases like MongoDB offer strong consistency options for critical operations like financial transactions.
- This ensures data integrity and reliability, which is essential for e-commerce applications where accurate order processing and inventory management are critical.

Why Other NoSQL Database Types Might Not Be Suitable for E-Commerce

1. Key-Value Stores:

- **Limited Data Modeling:** Key-value stores are not well-suited for storing complex, hierarchical data structures like products, customer information, and order details.

Example: Representing a product with multiple attributes (e.g., name, description, price, reviews) in a key-value store would require multiple key-value pairs, making it less efficient and harder to query.

2. Wide-Column Stores:

- **Rigid Schema:** Wide-column stores have a fixed schema, which can be limiting for e-commerce applications where data structures may need to evolve over time.

Example: Adding a new product attribute (e.g., color) would require modifying the schema in a wide-column store, which can be time-consuming and disruptive.

3. Graph Databases:

- **Focus on Relationships:** Graph databases are optimized for storing and querying interconnected data, which might not be ideal for the transactional nature of e-commerce applications.

Example: While graph databases can be used to model relationships between products and categories, they might not be as efficient for handling large volumes of transactional data and complex queries.