24CB404/24CM404/24CS404/24DS404/IT404

Hall Ticket Number:

II/IV B. Tech (Regular) DEGREE EXAMINATION

May, 2026 — **Common to CB, CM, CSE, DS & IT**

Fourth Semester — **Database Management Systems**

Time: Three Hours — **Maximum: 60 Marks**

Answer question 1 compulsorily. **(12X1 = 12 Marks)**

Answer one question from each unit. **(4X12 = 48 Marks)**

			CO	BL	M
1.	a)	Define the database.	CO1	L1	1M
	b)	Identify different actors involved in a database system environment.	CO1	L2	1M
	c)	State the concept of data independence in database systems.	CO1	L1	1M
	d)	Define the SELECT operation in relational algebra.	CO2	L2	1M
	e)	List different types of SQL constraints used in schema definition.	CO2	L2	1M
	f)	State the purpose of views in SQL.	CO2	L1	1M
	g)	Define functional dependency with an example.	CO3	L2	1M
	h)	State the concept of normalization in relational databases.	CO3	L1	1M
	i)	Identify the purpose of indexing in file structures.	CO3	L1	1M
	j)	Define a transaction in database systems.	CO4	L1	1M
	k)	State the concept of serializability in transaction processing.	CO4	L2	1M
	l)	Define concurrency control.	CO4	L1	1M
		Unit –I			
2.	a)	Explain the characteristics of the database approach in detail.	CO1	L2	6M
	b)	Describe the three-schema architecture with a neat diagram.	CO1	L3	6M
		(OR)			
3.	a)	Illustrate the ER model components including entities, attributes, and relationships.	CO1	L2	6M
	b)	Discuss weak entity types and their role in ER design with an example.	CO1	L3	6M
		Unit –II			
4.	a)	Explain basic relational algebra operations with suitable examples.	CO2	L2	6M
	b)	Describe tuple relational calculus with appropriate expressions.	CO2	L3	6M
		(OR)			
5.	a)	Illustrate SQL queries for INSERT, DELETE, and UPDATE operations.	CO2	L3	6M
	b)	Explain JOIN operations in SQL with different types and examples.	CO2	L2	6M
		Unit –III			
6.	a)	Construct the B tree to insert the following key values in order of 4 of the tree is 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42.	CO3	L3	6M
	b)	Describe second normal form and third normal form with examples.	CO3	L3	6M
		(OR)			
7.	a)	Discuss BCNF and its importance in database design.	CO3	L3	6M
	b)	Explain lossless join and dependency preserving decomposition with examples.	CO3	L2	6M
		Unit –IV			
8.	a)	Explain ACID properties of transactions with suitable examples.	CO4	L3	6M
	b)	Describe the transaction states with neat sketch.	CO4	L2	6M
		(OR)			
9.	a)	Discuss two-phase locking protocol and its variants with examples.	CO4	L3	6M
	b)	Explain timestamp ordering protocol with a suitable example.	CO4	L2	6M

1(a) Define the database.
A database is a collection of related data that is organized so that it can be easily accessed, managed, and updated. It represents some aspect of the real world and is designed for a specific purpose.
1(b) Identify different actors involved in a database system environment.
The main actors involved in a database system are:
1. Database Administrators (DBA)
2. Database Designers
3. System Analysts and Application Programmers
4. End Users
5. DBMS System Designers and Implementers

1(c) State the concept of data independence in database systems.
Data independence is the ability to modify the schema at one level of a database system without affecting the schema at the next higher level.
Types:
- Physical Data Independence
- Logical Data Independence

1(d) Define the SELECT operation in relational algebra.
The SELECT operation retrieves tuples from a relation that satisfy a given condition.
Notation: σ(condition)(Relation)
Example: σ(Dept='CSE')(STUDENT)
1(e) List different types of SQL constraints used in schema definition.
SQL constraints are:
1. PRIMARY KEY
2. FOREIGN KEY
3. UNIQUE
4. NOT NULL
5. CHECK
6. DEFAULT

1(f) State the purpose of views in SQL.
A view is a virtual table created from one or more tables. It is used for:
- Security
- Simplifying complex queries
- Data abstraction
- Customized data presentation

1(g) Define functional dependency with an example.
A functional dependency exists when one attribute uniquely determines another attribute.
Notation: $X \rightarrow Y$
Example: Student_ID → Student_Name
Here, Student_ID uniquely determines Student_Name.
1(h) State the concept of normalization in relational databases.
Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity.
1(i) Identify the purpose of indexing in file structures.
Indexing is used to improve the speed of data retrieval operations in a database.
1(j) Define a transaction in database systems.
A transaction is a sequence of database operations treated as a single logical unit of work.
Example: Bank money transfer.
1(k) State the concept of serializability in transaction processing.
Serializability ensures that concurrent execution of transactions produces the same result as serial execution.
1(l) Define concurrency control.
Concurrency control is the process of managing simultaneous execution of transactions without causing inconsistency.

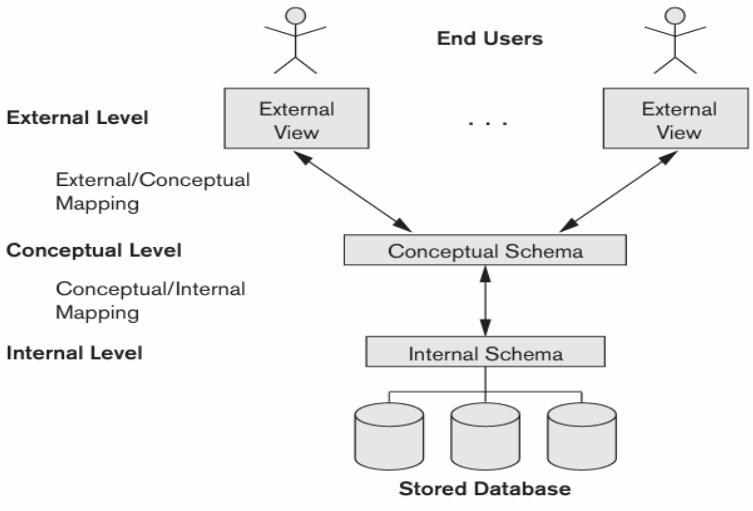
2a) Explain the characteristics of the database approach in detail.

S. No	Characteristic	Description
1	**Self-describing nature of a database system**	A database system stores **data along with metadata** (schema, constraints) in a **system catalog**,
2	**Insulation between programs and data (Data Independence)**	Application programs are **independent of data structure changes**. Changes in scema do not require rewriting application programs.
3	**Support for multiple views of the data**	Different users can have **different views** of the same database, showing only relevant data while hiding the rest.
4	**Sharing of data and multiuser transaction processing**	Multiple users can **access and update data simultaneously** with proper concurrency control and transaction management.
5	**Control of data redundancy**	A single database reduces duplicate data storage, minimizing redundancy and inconsistency compared to file systems.
6	**The of integrity constraints**	The DBMS enforces rules and constraints (key, domain, referential integrity) to maintain data accuracy and consistency.
7	**Security and authorization**	DBMS provides access control mechanisms to protect data from unauthorized access.
8	**Backup and recovery**	DBMS supports backup and recovery mechanisms to restore data after system failures.

2b) **Describe the three-schema architecture with a neat diagram.**

Three of the four important characteristics of the database approach, listed in Section 1.3, are (1) use of a catalog to store the database description (schema) so as to make it self-describing, (2) insulation of programs and data (program-data and program-operation independence), and (3) support of multiple user views. In this section we specify architecture for database systems, called the three-schema architecture, 9 that was proposed to help achieve and visualize these characteristics. Then we discuss the concept of data independence further.

Figure 2.2
The three-schema architecture.



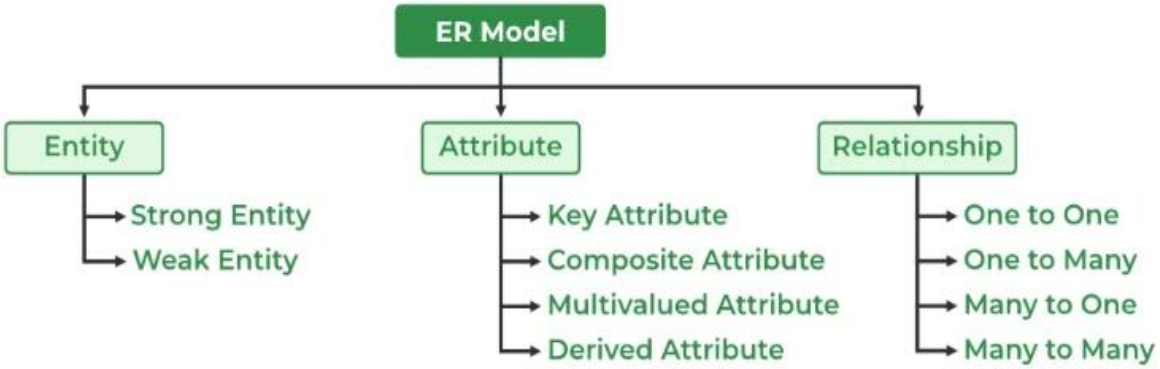
The **Three-Schema Architecture** The goal of the three-schema architecture, illustrated in Figure 2.2, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

1. The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

2. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.

3. The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

3a) **Illustrate the ER model components including entities, attributes, and relationships.**



- I. Entity: An Entity represents a real-world object, concept or thing about which data is stored in a database.
- Example of entities: Real-World Objects: Person, Car, Employee etc.
- Concepts: Course, Event, Reservation etc.
- Things: Product, Document, Device etc.
- Entity Set: An entity refers to an individual object of an entity type, and the collection of all entities of a particular type is called an entity set.
- For example, E1 is an entity that belongs to the entity type "Student," and the group of all students forms the entity set.
- **Types of Entity :**There are two main types of entities: **1. Strong Entity 2. Weak Entity**
- **1. Strong Entity :** A Strong Entity is a type of entity that has a key Attribute that can uniquely identify each instance of the entity.
- A Strong Entity does not depend on any other Entity in the Schema for its identification.
- **2. Weak Entity :** A Weak Entity cannot be uniquely identified by its own attributes alone.
- It depends on a strong entity to be identified.
- **Types of Attributes**
- **1. Key Attribute:** The attribute which uniquely identifies each entity in the entity set is called the key attribute.
- For example, Roll_No will be unique for each student.
- **2. Composite Attribute :**An attribute composed of many other attributes is called a composite attribute.
- For example, the Address attribute of the student Entity type consists of Street, City, State, and Country.
- **3. Multivalued Attribute :** A Multivalued Attribute in the ER (Entity–Relationship) model is an attribute that can have more than one value for a single entity.
- Definition (Exam-oriented): A multivalued attribute is an attribute that can store multiple values for one entity instance.
- Examples
- Phone_Number of a Person (a person can have many phone numbers)
- Email_ID of an Employee, Skills of a Student, Languages Known by a Person.
- III- Relationship Type and Relationship Set
- A Relationship Type represents the association between entity types.
- For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course.
- A set of relationships of the same type is known as a relationship set.
- Binary Relationship: When there are TWO entities set participating in a relationship, the relationship is called a binary relationship.

- Unary/Recursive Relationship: When there is only ONE entity set participating in a relation, the relationship is called a unary relationship.

1. One-to-One : When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one.

2. One-to-Many : In a one-to-many relationship, one entity can be associated with multiple entities.

3.Many-to-One : When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one.

4.Many-to-Many : When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that an employee can work on multiple projects and each project can have multiple employees working on it. So, the relationship will be many-to-many (M:N).

3b) Discuss weak entity types and their role in ER design with an example.

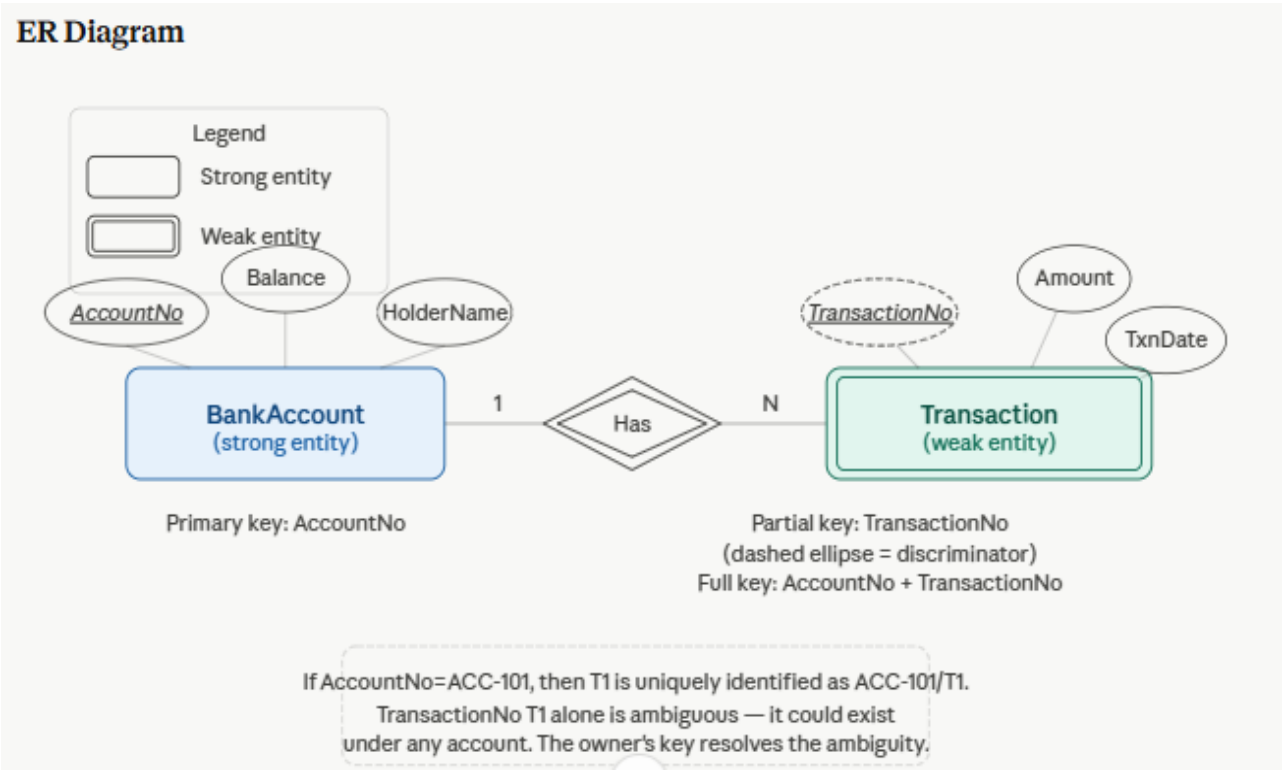
A **weak entity type** is an entity that **cannot be uniquely identified by its own attributes alone** — it depends on another entity (called the **owner** or **identifying entity**) for its existence and identification.

characteristics:

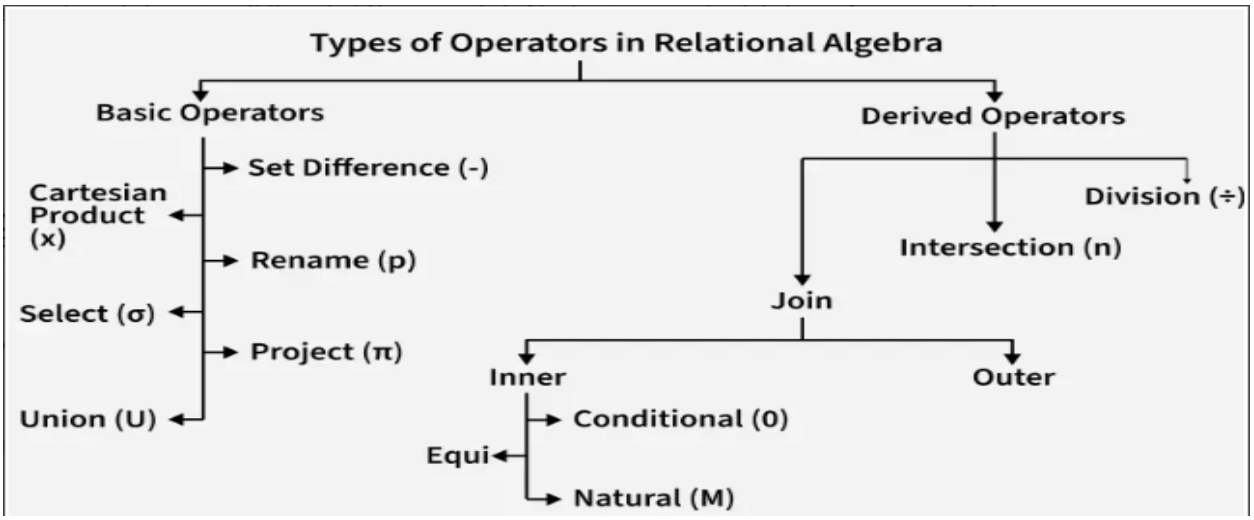
- Has no primary key of its own — only a **partial key** (discriminator)
- Existence depends on the **identifying (strong) entity**
- Connected to its owner via an **identifying relationship**
- Represented with a **double rectangle** in ER diagrams

Concept	Strong Entity	Weak Entity
Identification	Own primary key	Needs owner's key + partial key
Existence	Independent	Dependent on owner
ER Notation	Single rectangle	Double rectangle
Relationship	Regular diamond	Double diamond

Classic Example — Bank Account & Transaction



4A) Explain basic relational algebra operations with suitable examples.



1.Unary Relational Operations: SELECT and PROJECT.

- The **SELECT operator is unary**; that is, it is **applied to a single relation.**
- **The SELECT operation** is used to choose a **subset of the tuples** from a **relation that satisfies a selection condition**.
- One can **consider the SELECT operation** to be a **filter** that keeps **only those tuples that satisfy a qualifying condition and restrict the tuples** in a relation.
- In general, the SELECT operation is denoted by $\sigma_{<\text{selection condition}>}(R)$
- **For example:**
- **1.** to select the **EMPLOYEE** tuples whose **department is 4** with a SELECT operation..
- **Query : σ Dno=4 (EMPLOYEE)**

2. The PROJECT Operation:

- The **SELECT operation** chooses **some of the rows from the table** while **discarding other rows.**
- The **PROJECT operation**, on the other hand**, selects certain columns from the table** and **discards the other columns.**
- The result of the **PROJECT operation** can be **visualized as a vertical partition** of the relation into **two relations**.
- One has the **needed columns (attributes)** and contains the **result of the operation**, and the **other contains the discarded columns.**

The **PROJECT operation** removes any **duplicate tuples**, so the result of the PROJECT operation is a set of **distinct tuples,** This is **known as duplicate elimination.**

$$\pi_{<\text{attribute list}>}(R)$$

3.Sequences of Operations and RENAME Operation:
In general, the most of queries, we need to apply several relational algebra operations one after the other.
Either we can write the operations as a single relational algebra expression.
We can write a single relational algebra expression, also known as an in-line expression.
For example
To retrieve the first name, last name, and salary of all employees who work in department number 5.

$$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$$

4B) Describe tuple relational calculus with appropriate expressions

Tuple Relational Calculus (TRC)

Tuple Relational Calculus is a **non-procedural query language** for relational databases. Unlike relational algebra (which tells the database *how* to retrieve data), TRC tells the database *what* data to retrieve — the system figures out the how.

It is based on **first-order predicate logic** and was proposed by E.F. Codd as a formal foundation for query languages like SQL.

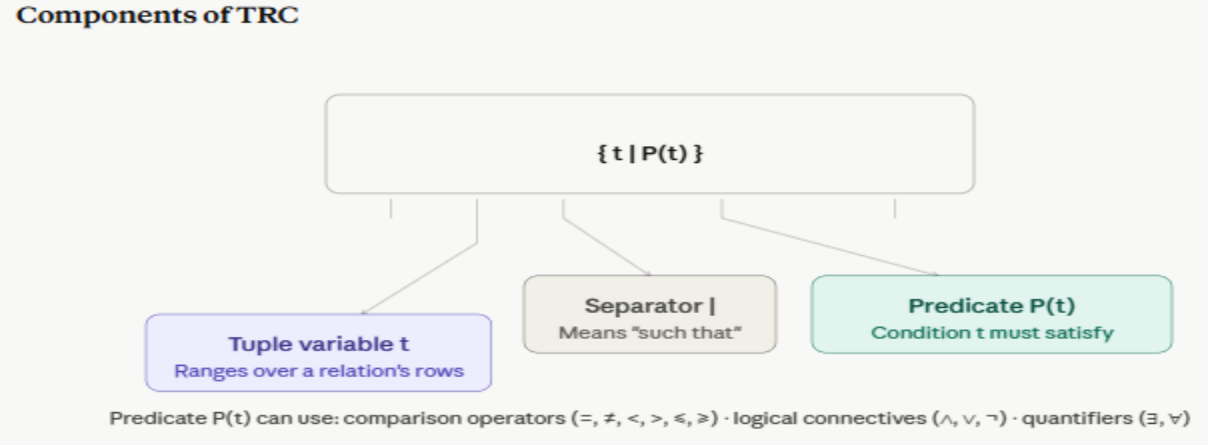
General form

The fundamental TRC expression has the form:

{ t | P(t) }

which reads as: *"the set of all tuples t such that predicate P(t) is true."*

- t is a **tuple variable** that ranges over a relation
- P(t) is a **predicate (condition)** that t must satisfy
- The result is the set of all tuples satisfying the predicate



Logical connectives and quantifiers used in TRC

TRC predicates are built using standard first-order logic operators:

Symbol	Name	Meaning
∧	AND (conjunction)	Both conditions must be true
∨	OR (disjunction)	At least one condition must be true
¬	NOT (negation)	Condition must be false
∃	Existential quantifier	There exists at least one tuple satisfying the condition
∀	Universal quantifier	For all tuples, the condition must hold
⟹	Implication	If ... then ...

5A) **Illustrate SQL queries for INSERT, DELETE, and UPDATE operations.**

SQL Data Manipulation: INSERT, DELETE, and UPDATE

Here is an interactive SQL playground showing all three DML operations — INSERT, DELETE, and UPDATE — with a live table that changes as you apply each operation. SQL (Structured Query Language) provides commands to manipulate data stored in database tables.

The commonly used data manipulation operations are:

1. INSERT 2. DELETE 3.UPDATE

These operations are part of **DML (Data Manipulation Language)**.

1. INSERT Operation

The **INSERT** command is used to add new records into a table.

Syntax

INSERT INTO table_name VALUES (value1, value2, value3);

Example Consider the STUDENT table:

Student_ID	Name	Department
101	Ravi	CSE

To insert a new record: INSERT INTO STUDENT VALUES (102, 'Sita', 'IT');

Result Student_ID	Name	Department
101	Ravi	CSE
102	Sita	IT

INSERT with Column Names

INSERT INTO STUDENT(Student_ID, Name, Department) VALUES (103, 'Rahul', 'ECE');

Advantages of INSERT

- Adds new records easily, Supports multiple row insertion, Maintains database growth

2. DELETE Operation

The **DELETE** command is used to remove records from a table.

Syntax : DELETE FROM table_name WHERE condition;

Example : To delete the student whose ID is 102:

DELETE FROM STUDENT WHERE Student_ID = 102;

Result Student_ID	Name	Department
101	Ravi	CSE
103	Rahul	ECE

DELETE All Records

DELETE FROM STUDENT; This removes all rows but keeps the table structure.

Importance of WHERE Clause

Without WHERE condition, all rows will be deleted.

Advantages of DELETE :

- Removes unwanted records ,Maintains accurate data ,Helps in database maintenance

3. UPDATE Operation

The **UPDATE** command is used to modify existing records in a table.

Syntax

UPDATE table_name SET column_name = value WHERE condition;

Example: To change department of Student_ID 101 to IT:

UPDATE STUDENT SET Department = 'IT' WHERE Student_ID = 101;

Result

Student_ID	Name	Department
101	Ravi	IT
103	Rahul	ECE

UPDATE Multiple Columns

UPDATE STUDENT SET Name = 'Ramesh', Department = 'CSE' WHERE Student_ID = 103;

Advantages of UPDATE

- Modifies existing data efficiently, Maintains latest information, Supports multiple column updates.

5b) Explain JOIN operations in SQL with different types and examples

1. **The JOIN Operation: The JOIN operation is Bowtie, denoted by It is used to combine related tuples from two relations into single "longer" tuples.**
2. **This operation is very important for any relational database with more than a single relation because it allows us to process relation ships among relations.**
3. The general form of a JOIN operation on two relations[5] $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$ is

   $$R \bowtie_{\text{<join condition>}} S$$
4. **The result of the JOIN is a relation Q with n + m attributes Q(A1,A2, ...,An,B1,B2, ... ,Bm) in that order; Q has one tuple for each combination of tuples— one from R and one from S —whenever the combination satisfies the join condition.**
5. **To illustrate JOIN, suppose that we want to retrieve the name of the manager of each department.**

6. **To get the manager's name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr_ssn value in the department tuple.**
7. **We do this by using the JOIN operation and then projecting the result over the necessary attributes.**
8. **The first operation is , Note that Mgr_ssn is a foreign key of the DEPARTMENT relation that references Ssn, the primary key of the EMPLOYEE relation.**
9. **This referential integrity constraint plays a role in having matching tuples in the referenced relation EMPLOYEE.**
10. **where each <condition> is of the form Ai θ Bj ,**
11. **Ai is an attribute of R, Bj is an attribute of S, Ai and Bj have the same domain, and θ(theta) is one of the comparison operators {=,<,<_,>,≥,≠}.**
12. **A JOIN operation with such a general join condition is called a THETA JOIN.**
13. **types of Join: 1. Inner Join: Inner Join is a join operation in DBMS that combines two or more tables based on related columns and returns only rows that have matching values among tables.it is classified into two types 1) Conditional join 2) Equi Join.**
14. **2. Outer Join Outer join is a type of join that retrieves matching as well as non-matching records from related tables. There are three types of outer join.**
15. **Left outer join. (It is also called left join. This type of outer join retrieves all records from the left table and retrieves matching records from the right table.)**
16. **Right outer join. (It is also called a right join. This type of outer join retrieves all records from the right table and retrieves matching records from the left table.)**
17. **Full outer join. (FULL JOIN creates the result set by combining the results of both LEFT JOIN and RIGHT JOIN. The result set will contain all the rows from both tables.).**

6a) **Construct the B tree to insert the following key values in order of 4 of the tree is 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42.**

B-Tree Construction (Order 4) In a B-tree of order 4:

- Each node can hold a **maximum of 3 keys** (order − 1)
- Each node can have a **maximum of 4 children**
- When a node overflows (gets 4 keys), it **splits** — the middle key promotes up
- The **split point** is at position [4/2] = 2nd key (the median promotes)

I'll first work through every insertion step precisely, then display the full step-by-step stepper.

Step-by-step trace (order 4 → max 3 keys, split on 4th key, median promotes):

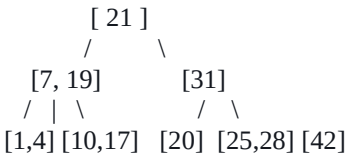
- Insert 1, 4, 7 → root: [1, 4, 7]
- Insert 10 → overflow [1,4,7,10], median=7 promotes → root:[7], left:[1,4], right:[10]
- Insert 17 → goes right → [10,17]
- Insert 21 → goes right → [10,17,21]
- Insert 31 → right node [10,17,21,31] overflows, median=21 promotes → root:[7,21], left:[1,4], mid:[10,17], right:[31]
- Insert 25 → goes to right:[25,31]
- Insert 19 → goes to mid:[10,17,19]
- Insert 20 → goes to mid:[10,17,19,20] overflows, median=19 promotes → root:[7,19,21], left:[1,4], c2:[10,17], c3:[20], c4:[25,31]
- Insert 28 → goes to c4:[25,28,31]
- Insert 42 → goes to c4:[25,28,31,42] overflows, median=31 promotes → root:[7,19,21,31] overflows, median=21 promotes → new root:[21], left subtree root:[7,19], right subtree root:[31] → left:[1,4],[10,17],[20] right:[25,28],[42]

Complete insertion trace (summary table)

Step	Key	Action	Split?	Root after
1	1	Insert into empty tree	No	[1]
2	4	Insert into root	No	[1, 4]
3	7	Insert into root — now full	No	[1, 4, 7]
4	10	Overflow → **7 promotes**	Yes	[7]

Step	Key	Action	Split?	Root after
5	17	Goes to right child	No	[7]
6	21	Goes to right child	No	[7]
7	31	Right child overflows → **21 promotes**	Yes	[7, 21]
8	25	Goes to rightmost child	No	[7, 21]
9	19	Goes to middle child	No	[7, 21]
10	20	Middle child overflows → **19 promotes**	Yes	[7, 19, 21]
11	28	Goes to rightmost child	No	[7, 19, 21]
12	42	Leaf overflows → **31 promotes** → root overflows → **21 promotes** to new root	Double split	[21]

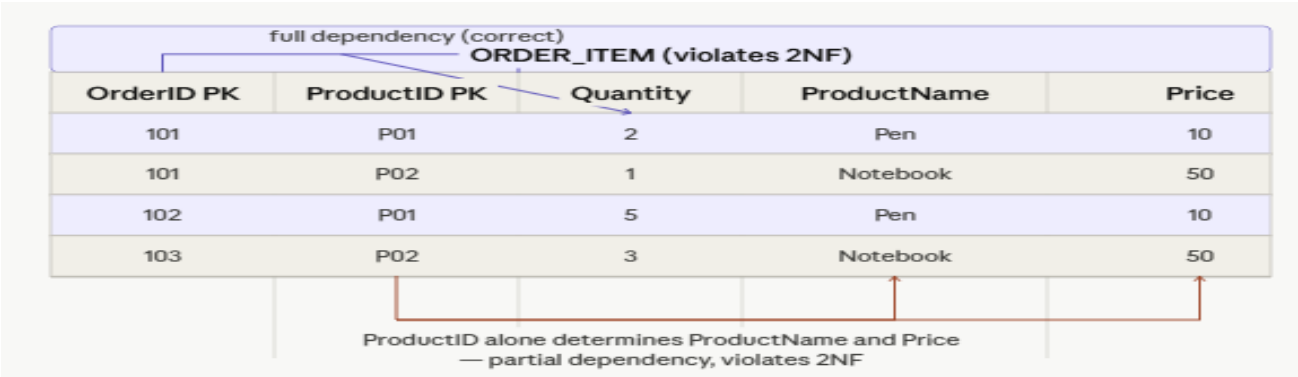
Final B-tree structure (order 4)

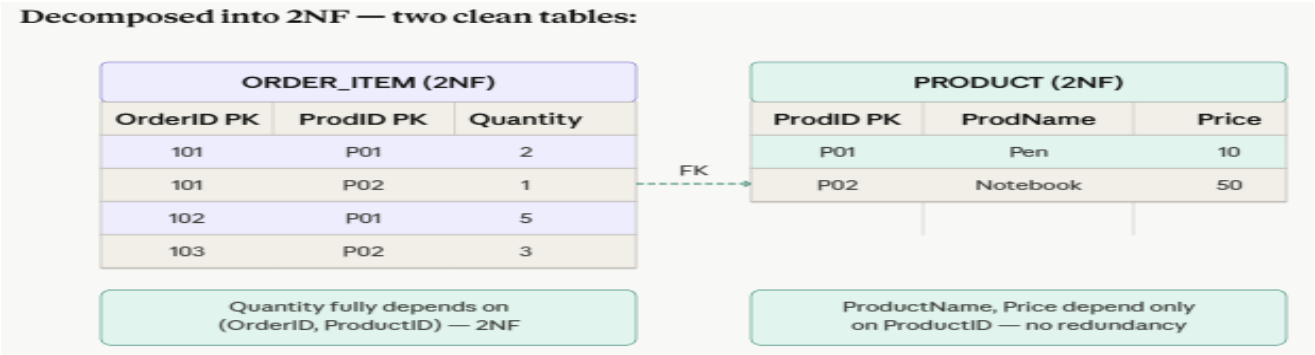


6b) Describe second normal form and third normal form with examples.

1. These two normal forms progressively eliminate different types of redundancy from relational databases. 2NF removes **partial dependencies**; 3NF removes **transitive dependencies**.
2. Second Normal Form (2NF) A relation is in 2NF if: It is already in **1NF** (atomic values, no repeating groups), and every non-key attribute is **fully functionally dependent** on the entire primary key — not just a part of it.
3. This only becomes relevant when the primary key is **composite** (made of two or more attributes). A partial dependency means a non-key attribute depends on only *part* of the composite key, not the whole thing.

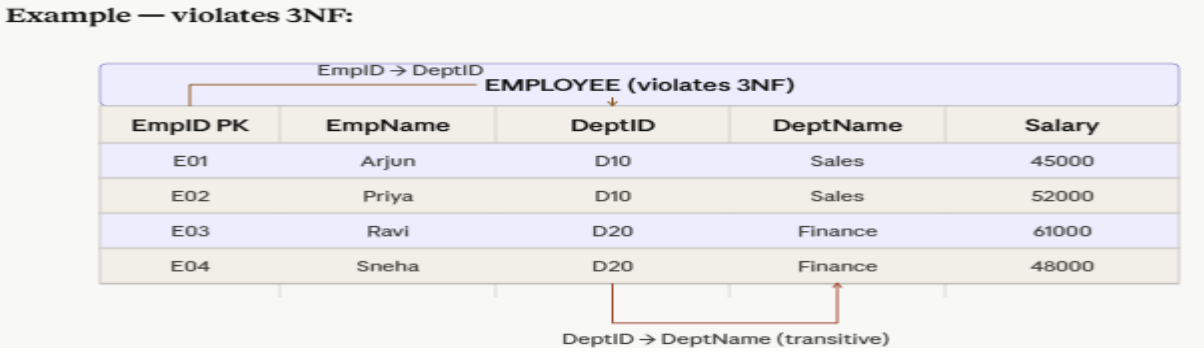
Example — violates 2NF:

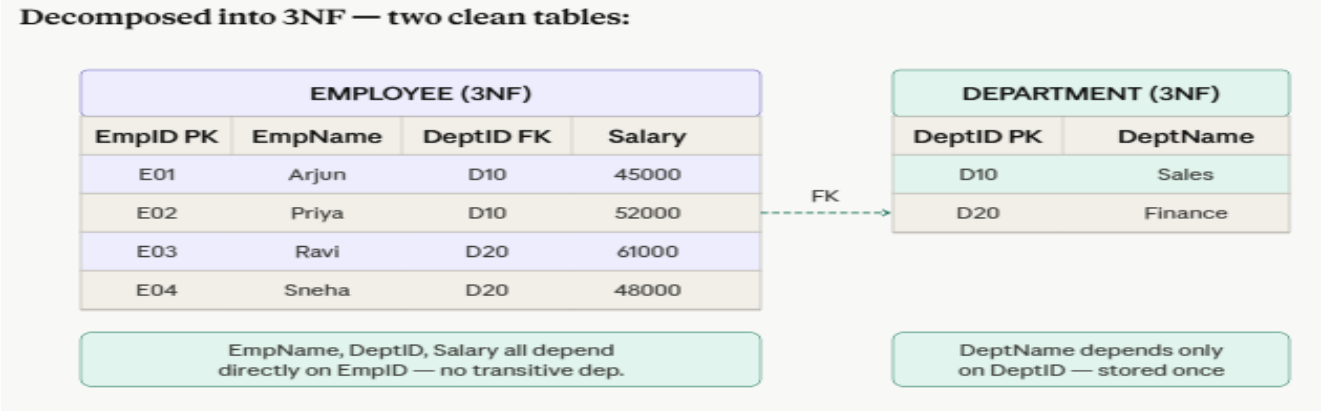




Third Normal Form (3NF) A relation is in 3NF if:

1. It is already in **2NF**, and No non-key attribute is **transitively dependent** on the primary key — meaning no non-key attribute determines another non-key attribute.
2. A transitive dependency occurs when: Primary Key → A → B (B depends on A, not directly on the key).





7A) Discuss BCNF and its importance in database design.

1. BCNF (also called 3.5NF) is a slightly stronger version of Third Normal Form (3NF), introduced by Raymond Boyce and Edgar Codd to address certain anomalies that 3NF doesn't fully eliminate.
2. Formal Definition :A relation R is in BCNF if and only if **for every non-trivial functional dependency X → Y, X must be a superkey** — meaning X must uniquely identify every tuple in the relation.
3. 3NF allows a non-prime attribute to be determined by something that isn't a superkey, as long as it's part of a candidate key. BCNF eliminates even that exception.
4. **The rule:** In 3NF, a dependency A → B is allowed if B is a prime attribute (part of a candidate key). BCNF removes this allowance entirely.
5. A Classic Example Consider a relation: Enrollment(Student, Course, Instructor)With these constraints: Each instructor teaches only one course, Each student-course pair has one instructor.
6. Anomalies BCNF Prevents:

- **Update anomaly** — Changing an instructor's course requires updating multiple rows, risking inconsistency.
- **Insertion anomaly** — You can't record that an instructor teaches a course unless a student is enrolled.
- **Deletion anomaly** — Deleting the last student in a course also deletes the instructor–course mapping.

Importance in Database Design

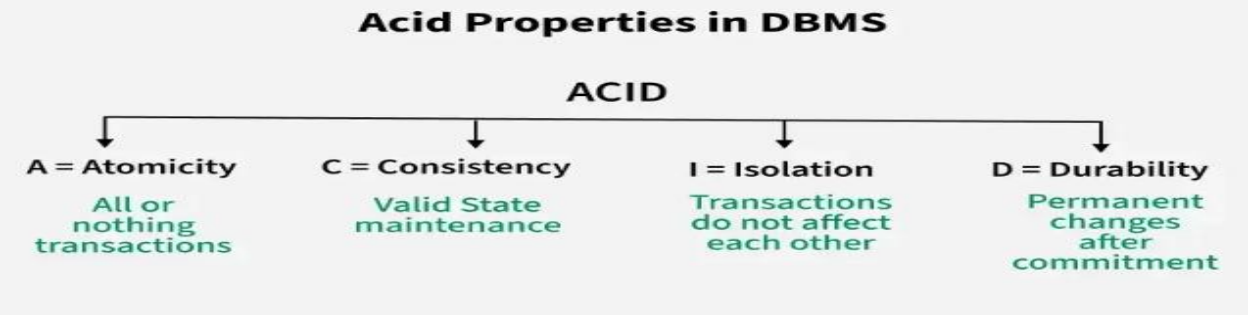
1. Eliminates Redundancy More Thoroughly BCNF ensures no fact is stored more than once due to a non-superkey determinant, reducing storage waste and inconsistency.

2. Guarantees Update Consistency Since every functional dependency flows from a superkey, updating a value requires changing it in exactly one place.

3. Stronger Integrity Foundation It enforces a cleaner, more principled structure where each table has a clear, singular identity.

4. Basis for Further Normalization BCNF is a prerequisite stepping stone toward 4NF and 5NF, which handle multi-valued and join dependencies.

8a) **Explain ACID properties of transactions with suitable examples.**

1. **ACID Properties of Transactions:** ACID is an acronym for **Atomicity, Consistency, Isolation, and Durability** — the four fundamental properties that guarantee database transactions are processed reliably, even in the face of errors, crashes, or concurrent access.
2. Transactions are **fundamental operations** that **allow us to modify and retrieve data.**
3. it is important that these transactions are executed in a way that maintains **consistency, correctness**, and **reliability** even in the **case of failures/errors**. This is where the **ACID properties come** into play.
4. 

5. **Atomicity:** "All or nothing" **— a transaction either completes fully or has no effect at all. If any step fails, all previous steps in that transaction are rolled back.**

6. **Example — Bank Transfer**

```
BEGIN TRANSACTION;
    UPDATE accounts SET balance = balance - 5000 WHERE id = 'A';  -- Debit A
    UPDATE accounts SET balance = balance + 5000 WHERE id = 'B';  -- Credit B
COMMIT;
```

7. If the system crashes after debiting A but before crediting B, atomicity ensures the debit is **rolled back** — money is never lost in limbo. Either both updates happen, or neither does.

8. **Consistency:** "Valid state in, valid state out" **— a transaction must take the database from one consistent state to another, respecting all defined rules, constraints, and integrity conditions.**

9. **Example — Account Balance Constraint: Suppose a rule states: *account balance cannot go below 0.* BEGIN TRANSACTION;**

```
    UPDATE accounts SET balance = balance - 10000 WHERE id = 'A';
        -- A has only ₹3000. This violates the constraint!
        ROLLBACK;  -- Transaction aborted to preserve consistency
```

10. The database refuses the transaction rather than allow an invalid state (negative balance) to persist.

Consistency is partly the DBMS's job (enforcing constraints) and partly the **application's responsibility** (writing logically correct transactions).

11. Isolation: **"Concurrent transactions don't interfere"** — even when multiple transactions run simultaneously, each one behaves as if it were the only transaction running.
12. Example — Concurrent Ticket Booking ,Two users try to book the **last available seat** at the same time:

Time	T1 (User A)	T2 (User B)
t1	Read seats = 1	
t2		Read seats = 1
t3	Book seat, seats = 0	
t4		Book seat, seats = 0

4. Durability: "Committed data survives" **— once a transaction is committed, its changes are permanent, even if the system crashes immediately afterward.**

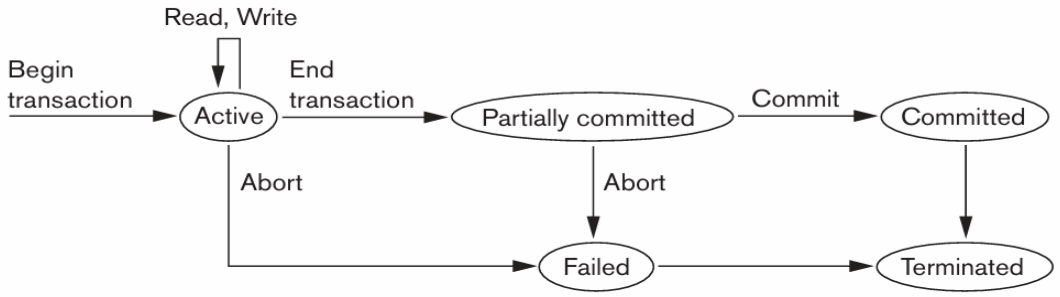
Example — Order Confirmation

```
BEGIN TRANSACTION;
  INSERT INTO orders (id, item, status) VALUES (101, 'Laptop', 'Confirmed');
  UPDATE inventory SET stock = stock - 1 WHERE item = 'Laptop';
COMMIT;  ← Power failure happens RIGHT HERE
```

After recovery, the database must still reflect the committed order. The customer's purchase is not lost.

8b) Describe the transaction states with neat sketch.

Figure 21.4
State transition diagram illustrating the states for transaction execution.



[5]Optimistic concurrency control (see Section 22.4) also requires that certain checks are made at this point to ensure that the transaction did not interfere with other executing transactions.

- A transaction is an atomic unit of work that should either be completed in its entirety or not done at all.
- For recovery purposes, the system needs to keep track of when each transaction starts, terminates, and commits or aborts.
- BEGIN_TRANSACTION. This marks the beginning of transaction execution.
- ■ READ or WRITE. These specify read or write operations on the database items that are executed as part of a transaction.
- ■ END_TRANSACTION. This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution.
- COMMIT_TRANSACTION. This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
- ROLLBACK (or ABORT). This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

The Five Transaction States

1. Active

This is the **initial state** of every transaction. The transaction enters this state the moment it begins and stays here while it is executing — performing reads and writes on the database. The self-loop (Read, Write) on this state indicates it can keep executing multiple operations before moving on.

2. Partially Committed

After the **last statement** of the transaction executes (End Transaction), it moves to this state. The transaction has finished its work but the changes have **not yet been permanently written** to disk. Final checks (like constraint validation and concurrency checks) happen here.

- If everything is valid → moves to **Committed,** If something fails (e.g., a constraint is violated, system error) → moves to **Failed**

3. Committed

The transaction has **successfully completed** and all its changes are permanently saved to the database. This is the desired outcome. From here, the transaction moves to Terminated.

This is where **Durability** (the D in ACID) kicks in — once committed, data survives crashes.

4. Failed

The transaction cannot proceed normally. This happens when:

- An **abort is explicitly issued** (from the Active state), or A **check fails** after partial commitment (e.g., constraint violation, deadlock)

A failed transaction must be **rolled back** — all its partial changes are undone to restore the database to a consistent state.

5. Terminated

The **final state** reached from both Committed and Failed. The transaction has ended its life cycle — either successfully or after a rollback. The system can now release all locks and resources held by this transaction.

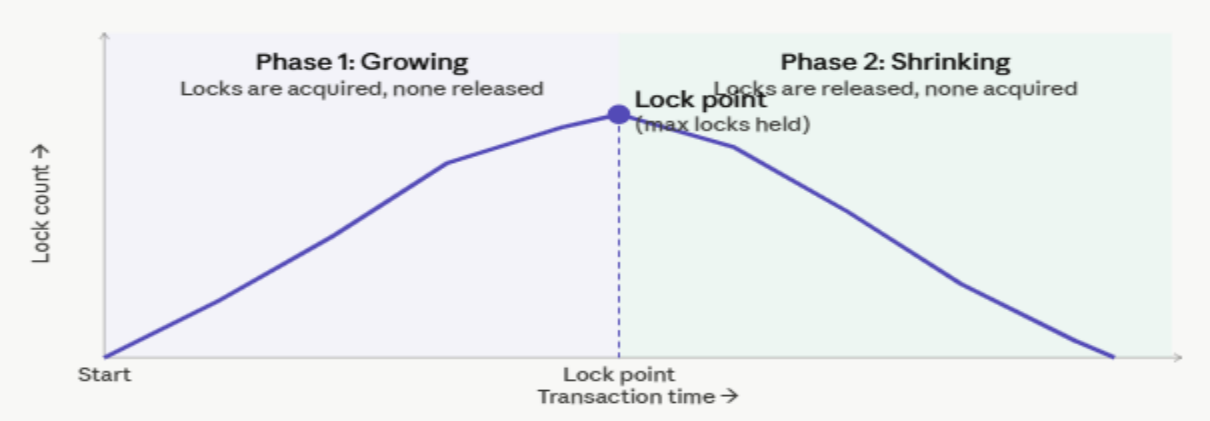
9a) **Discuss two-phase locking protocol and its variants with examples:**

Two-Phase Locking (2PL) Protocol

- Two-Phase Locking is a concurrency control protocol that ensures **serializability** in database transactions. It governs how and when transactions acquire and release locks, dividing every transaction's lifecycle into exactly two phases.
- Two-Phase Locking Protocol states that: A transaction must acquire all required locks before releasing any lock. The protocol divides transaction execution into two phases:

- **1) Growing Phase**, Shrinking Phase. Growing Phase, In this phase: Transaction can acquire locks, Transaction cannot release locks.
- Operations Allowed, Lock acquisition → Allowed, Unlock operation → Not allowed.

3. Example: Transaction T1:Lock(A)
   Read(A)
   Lock(B)
   Write(B)
4. Shrinking Phase: In this phase: Transaction can release locks, Transaction cannot acquire new locks, Operations Allowed, Unlock operation → Allowed, New lock acquisition → Not allowed.
5. Example, Unlock(A)
   Unlock(B)



9b) Explain timestamp ordering protocol with a suitable example.

- The **Timestamp Ordering Protocol** is a concurrency control method used in DBMS to ensure **serializability** of transactions without using locks.
- In this protocol, each transaction is assigned a unique timestamp when it starts execution. Transactions are executed according to the order of their timestamps.
- Timestamp Ordering Protocol is a concurrency control technique in which transactions are ordered based on their timestamps to avoid conflicts and maintain database consistency.
  Every transaction receives a unique timestamp.
  Older transactions get smaller timestamps.
  The DBMS uses timestamps to decide the execution order of read and write operations.
  Example :

Transaction	Timestamp
T1	10
T2	20

1. Read Timestamp RTS(X): Largest timestamp of any transaction that successfully read X.
2. Write Timestamp WTS(X): Largest timestamp of any transaction that successfully wrote X.
1. Read Operation Rule. Transaction Ti can read data item X only if:
   $TS(Ti) \geq WTS(X)$
2. Write Operation Rule. -Transaction Ti can write data item X only if:
   $TS(Ti) \geq RTS(X)$
   and
   $TS(Ti) \geq WTS(X)$

Name of the Subject : Database Management system.

Subject code : 24CS/IT/CB/DS/CM404

Signature of the Faculty:

1.
2.
3.
4.

HOD (DS)