**DATABASE MANAGEMENT SYSTEMS – BCS403**
**FOURTH SEMESTER B.E. EXAMINATION, JUNE/ JULY 2025**
**VTU EXAM SOLUTION**

# Module-1

**Q.1**

**a.** Explain the types of attributes with example.

| Attribute Type | Description | Example |
|---|---|---|
| Simple (Atomic) | Cannot be divided further | Age, Name |
| Composite | Can be divided into subparts | Name → First, Last |
| Derived | Computed from other attributes | Age (from DOB) |
| Multi-valued | Multiple values for a single entity | Phone_Numbers, Emails |
| Single-valued | Only one value for each entity | Roll_Number |
| Stored | Physically stored in DB | Date_of_Birth |
| Key | Uniquely identifies each entity | Student_ID, Emp_ID |

**(4 Marks)**

**b. Define database. Explain the main characteristics of the database approach.**

Definition - 2Marks

Characteristics - 3Marks

Database is a collection of related data. By data, we mean known facts that can be recorded andthat have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know.

A database has the following implicit properties:

▪ It represents some aspect of the real world, sometimes called the miniworldor the universe

of discourse (UoD). Changes to the miniworld are reflected in the database.

▪ It is a logically coherent collection of data, to which some meaning can be attached.

▪ It is designed, built, and populated with data for a specific purpose.It has an intended group

of users and some preconceived applications in which these users are interested.

**characteristics of the database approach**

The main characteristics of the database approach versus the file-processing approach are the following:

1. Self-describing nature of a database system

2. Insulation between programs and data

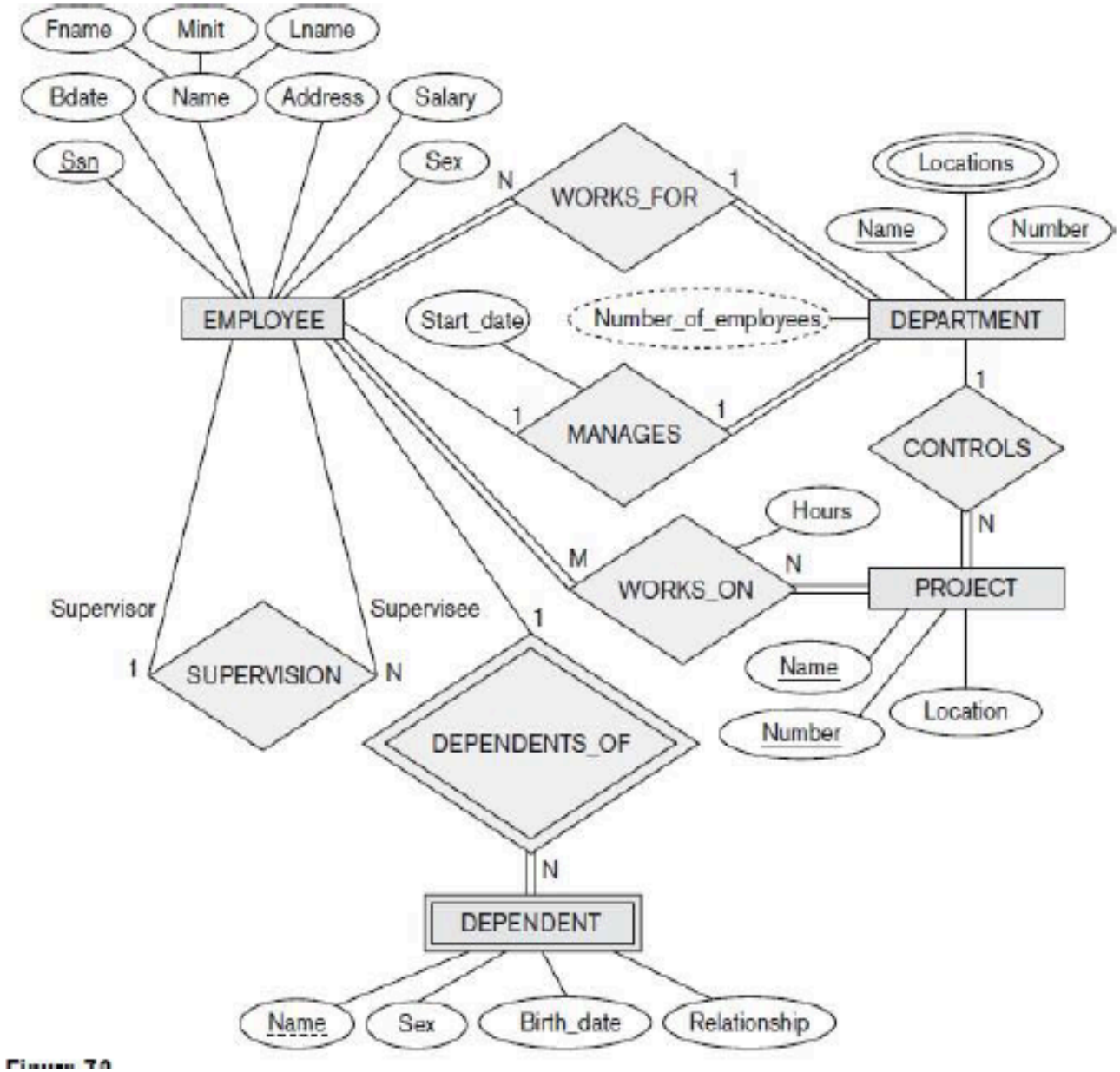
3. data abstraction

4. Support of multiple views of the data

5. Sharing of data and multiuser transaction processing

**(4 Marks)**

**c.** Show the ER diagram for an EMPLOYEE database by assuming your own entities (minimum 4) attributes and relationships, mention cardinality ratios wherever appropriate. **(8 Marks)**



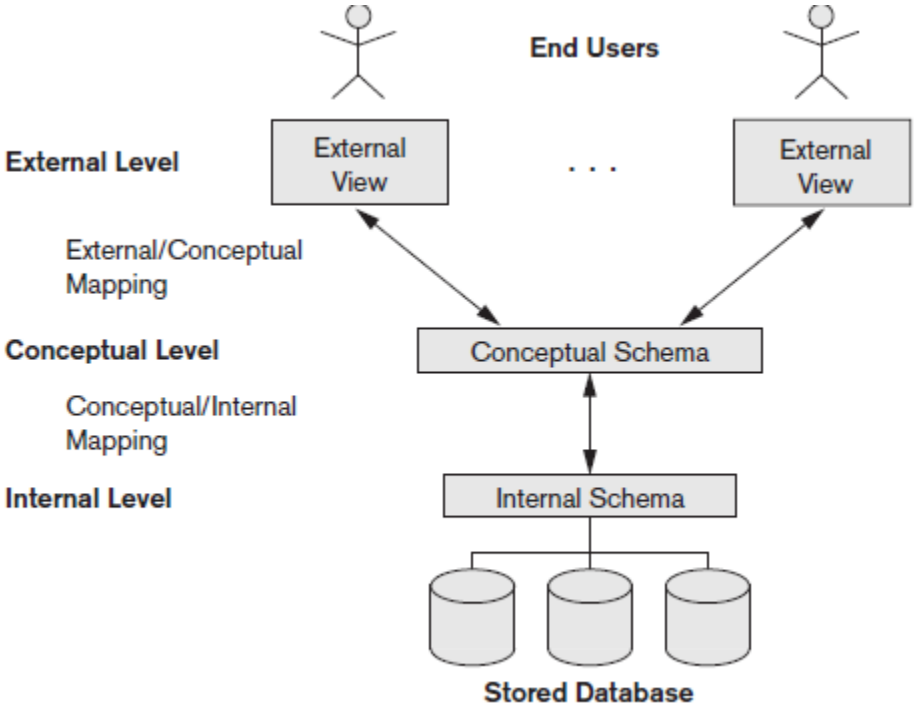

**Q.2**

a. Describe the three schema architecture. **(4 Marks)**

## The Three-Schema Architecture

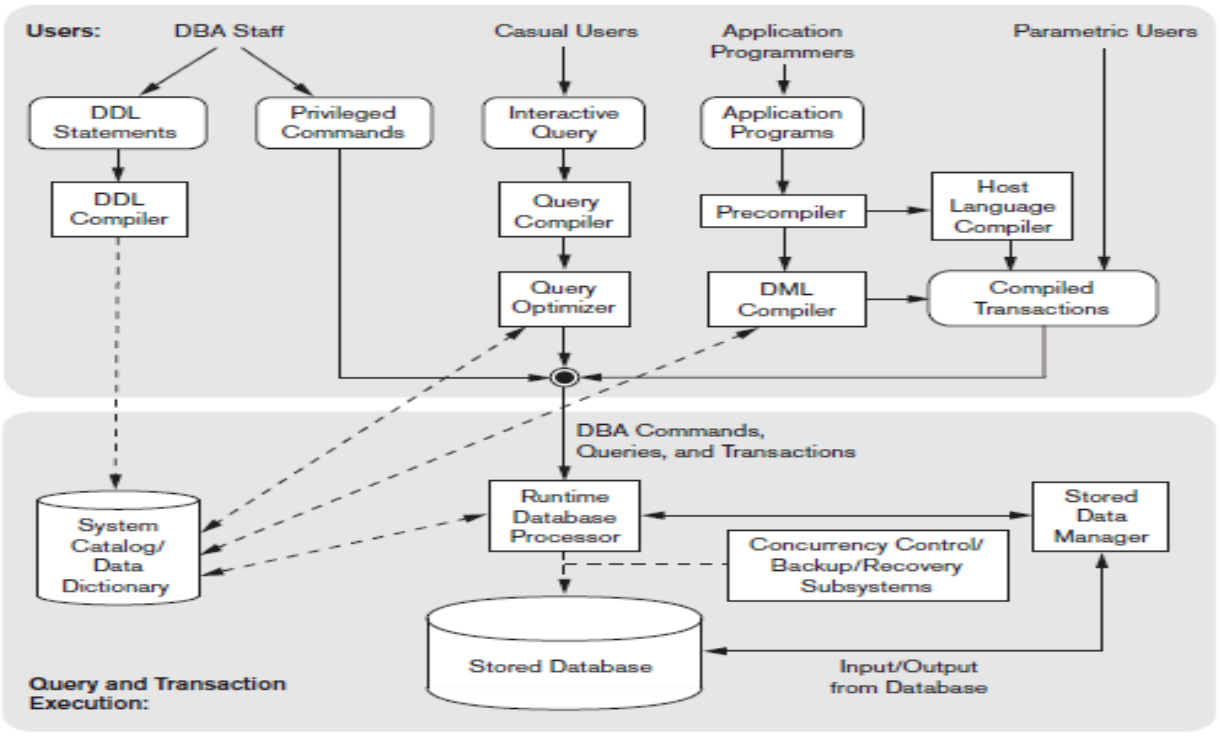
The goal of the three-schema architecture is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

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.
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. Each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.



b. Explain the component models of DBMS and their interaction with the help of diagram.

**(4 Marks)**




**The DBMS system structure is divided into two component modules.**

**Upper module** - different users of the database environment and their interfaces.

**Lower module** - internals of the DBMS responsible for storage of data and processing of transaction.

## UPPER Module

DBA staff

- Defines the database and makes changes to its description using the DDL and other privileged commands.
- The DBMS catalog stores the description of the schemas that are processed by the DLL compilers.

- The catalog includes names and sizes of the files, data types, storage details of each file, mapping information among schemas and constraints.

**Casual users (persons with occasional need)**

- Utilize menu based or form based query interfaces.
- The query compiler parses the queries, analysis the data elements for its correctness and converts it into internal form.
- The internal query is passed through the query optimizer for rearrangement of operations and elimination of redundancies.

**Application programmer**

- Writes programs in host languages.
- The precompiler separates the DML command and the host program.
- DML commands are submitted to the DML compiler and the rest of the program to the host compiler.
- The outputs of DML compiler and host compiler are linked by the compiled transaction to form the executable codes which includes the calls to the runtime database processor.

**Parametric users**

Supply the parameters (eg. account number and amount for bank withdrawal) using the compiled transactions.

The privileged commands, executable query plan and the compiled transactions with the runtime parameters are executed by the runtime database processor. It refers the data dictionary for updations.

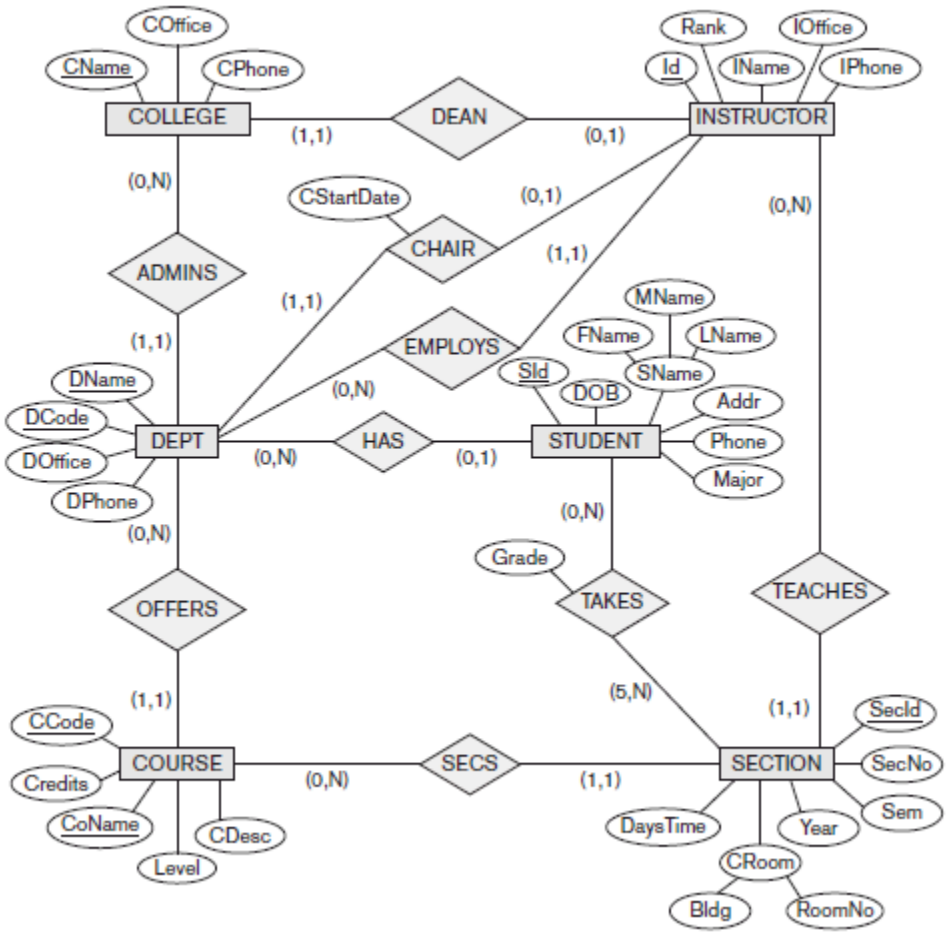
## LOWER module

Stored data manager carry out the input and output operations between the disk and main memory and provides support for database management including management of access paths and indexes.

- The file manager deals with the data structure options for storing in the database and maintains the metadata as well as indexes for various files.
- The interfacing between the main memory and disk is handled by the buffer manager.
- The transaction manager is responsible for concurrency and crash recovery by maintaining a record of all changes to the database.

c. Design ER diagram for a university database by assuming your own entities (4). Mention primary key constraints and relationships.

**(8 Marks)**




## Module-2

**Q.3**

a. Explain relational model constraints.

**(6 Marks)**

Relational ModelConstraints on databases can generally be divided into three main categories:
1. Constraints that are inherent in the data model known as **inherent model-based constraints or implicit constraints**.
2. Constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL known as **schema-based constraints or explicit constraints**.
**3.** Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs or in some other way known **as application-based or semantic constraints or business rules.**

The schema-based constraints include **domain constraints, key constraints, constraints on NULLs, entity integrity constraints, and referential integrity constraints.**

1. **Domain Constraints**

- ✓ Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A).
- ✓ The data types associated with domains typically include standard numeric data types for integers and real numbers. Characters, Booleans, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and other special data types.

2. **Key Constraints and Constraints on NULL Values**

- ✓ In the formal relational model, a relation is defined as a set of tuples.
- ✓ By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct.
- ✓ This means that no two tuples can have the same combination of values for all their attributes. Usually, there are other subsets of attributes of a relation schema R with the property that no two tuples in any relation state r of R should have the same combination of values for these attributes.
- ✓ Suppose that we denote one such subset of attributes by SK; then for any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that:
  $t1[SK] \neq t2[SK]$

3. **Relational Databases and Relational Database Schemas**
   - ✓ A relational database is a collection of many relations.
   - ✓ A relational database schema S is a set of relation schemas S = {R1, R2, … , Rm} and a set of integrity constraints IC.
   - ✓ A relational database state DB of S is a set of relation states DB = {r1, r2, … ,rm} such that each ri is a state of Ri and such that the ri relation states satisfy the integrity constraints specified in IC.
   - ✓ A relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}.
   - ✓ When we refer to a relational database, we implicitly include both its schema and its current state. A database state that does not obey all the integrity constraints is called **not valid**, and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a **valid state.**

4. **Entity Integrity, Referential Integrity, and Foreign Keys**
   - ✓ The entity integrity constraint states that **no primary key value can be NULL**. This is because the primary key value is used to identify individual tuples in a relation.
   - ✓ Key constraints and entity integrity constraints are specified on individual relations.
   - ✓ The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations.
   - ✓ Informally, the referential integrity constraint states that a **tuple in one relation that refers to another relation must refer to an existing tuple in that relation.**
   - ✓ For example, the attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.
   - ✓ The conditions for a **foreign key**, given below, specify a referential integrity constraint between the two relation schemas R1 and R2.

**Other types of constraints**

- ✓ The salary of an employee should not exceed the salary of the employee's supervisor and the maximum number of hours an employee can work on all projects per week is 56. Such constraints can be specified and enforced within the application programs that update the database, or by using a general-purpose constraint specification language. Sometimes called as **Semantic Integrity constraint**.

b. Explain the characteristics of relations with suitable example for each.

**(6 Marks)**

**Characteristics of Relations**

1. **Ordering of Tuples in a Relation**
   - ✓ A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them; hence, tuples in a relation do not have any particular order.
   - ✓ Similarly, when tuples are represented on a storage device, they must be organized in *some* fashion, and it may be advantageous, from a performance standpoint, to organize them in a way that depends upon their content.

2. **Ordering of Values within a Tuple**
   - ✓ The order of attributes and their values is not that important as long as the correspondence between attributes and values is maintained.
   - ✓ A tuple can be considered as a set of (<attribute>,<value> ) pairs, where each pair gives the value of the mapping from an attribute Ai to a value vi from dom(Ai). The ordering of attributes is not important, because the attribute name appears with its value.

3. **Values and NULLs in the Tuples**
   - ✓ Each value in a tuple is an atomic value; that is, it is not divisible into components.
   - ✓ An important concept is NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple.
   - ✓ NULL values has several meanings, such as **value unknown**, **value exists but is not available**, or **attributedoes not apply to this tuple.**

4. **Interpretation (Meaning) of a Relation**
   - ✓ Each tuple in the relation can then be interpreted as a **fact** or a particular instance of the assertion.
   - ✓ Each relation can be viewed as a **predicate** and each tuple in that relation can be viewed as an assertion for which that predicate is satisfied (i.e., has value **true**) for the combination of values in it.
   - ✓ Example:There exists a student having name Benjamin Bayer, having SSN 305-61-2435, having age 19, etc

c. Considering the following schema:

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

Write a relational algebra queries for the following:

i) Find the names of sailors, who have reserved red and a green boat
ii) Find the names of sailors who have reserved a red boat
iii) Find the names of sailors who have reserved a red or green boat
iv) Find the names of sailors who have reserved all boats

**(8 Marks)**

i)RedBoats ← π[bid](σ[color='red'](Boats))

GreenBoats ← π[bid](σ[color='green'](Boats))

RedSailors ← π[sid](Reserves ⋈ RedBoats)

GreenSailors ← π[sid](Reserves ⋈ GreenBoats)

Answer ← π[sname]((RedSailors ∩ GreenSailors) ⋈ Sailors)

ii)RedBoats ← σ[color='red'](Boats)

Answer ← π[sname]((Reserves ⋈ RedBoats) ⋈ Sailors)

iii)RedBoats ← σ[color='red'](Boats)

GreenBoats ← σ[color='green'](Boats)

RedGreenBoats ← RedBoats ∪ GreenBoats

Answer ← π[sname]((Reserves ⋈ RedGreenBoats) ⋈ Sailors)

iv)AllBoats ← π[bid](Boats)

SailorBoat ← π[sid, bid](Reserves)

Answer ← π[sname]((SailorBoat ÷ AllBoats) ⋈ Sailors)

OR

**Q.4**

a. Explain the steps to convert the basic ER model to relational Database schema.

**(6 Marks)**

b. Explain Unary relational operations with example.

**(6 Marks)**

c.

Consider the relation schema Employee database.

EMPLOYEE (Fname, Minit, Lname, SSN, Bdate, Address, Sex, Salary, Super_SSN, Dno)

DEPARTMENT (Dname, Dnumber, Mgr_SSN, Mgr_start_date)

PROJECT (Pname, Pnumber, Plocation, Dnum)

WORKS_ON (Essn, Pno, Hours)

DEPENDENT (Essn, Dependent_name, Sex, Bdate, Relationship)

Write relational algebra queries for the following:

i) Retrieve the name and address of all employees who work for the 'research' department.

ii) List the names of all employees with 2 or more dependents.

iii) Find the names of employees who work on all the projects controlled by department number

iv) List the names of employees who have no dependents.

**( 8 Marks)**

i)Result ← π Fname, Minit, Lname, Address (

σ Dname='Research' (DEPARTMENT) ⋈ EMPLOYEE.Dno = DEPARTMENT.Dnumber (EMPLOYEE)

)

ii)Dependent_Count ← γ Essn, COUNT(Dependent_name)→DepCount (DEPENDENT)

Result ← π Fname, Minit, Lname (

(σ DepCount ≥ 2 (Dependent_Count)) ⋈ EMPLOYEE.SSN = Dependent_Count.Essn

)

iii)Proj_D ← π Pnumber (σ Dnum = 5 (PROJECT))

Emp_Proj ← π Essn, Pno (WORKS_ON)

Result ← π Fname, Minit, Lname (

EMPLOYEE ⋈ SSN = Essn (

(γ Essn (Emp_Proj ÷ Proj_D))

)

)

iv)Emp_No_Dep ← π SSN (EMPLOYEE) − π Essn (DEPENDENT)

Result ← π Fname, Minit, Lname (

σ SSN ∈ Emp_No_Dep (EMPLOYEE)

)

## Module – 3

**Q.5**

**a. What is the need for normalization? Explain second and third normal form with example.**

**(6 Marks)**

**Answer:**

**Normalization of data** can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies.

The normalization procedure provides database designers

- A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree

**2NF:**

- A second normal is a method of arranging attributes semantically (logically) based on the constraints 1) a relation must be in first normal form and 2) relation should not contain any partial dependency.
- No non-prime attribute (attribute which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
- The partial dependency - is the proper subset of candidate key determines non-primary attribute in a relation.
- Every non-key attribute is fully functionally dependent on the primary key. Thus, no non-key attributes are functionally dependent on the part (but not all) of the primary key. That means, no partial dependency exists.
- Note: If a key is single attribute, then it is always in 2nd Normal form.

**2NF - Decomposition**

1. If a data item is fully functionally dependent on only a part of the primary key, move that data item and that part of the primary key to a new table.
2. If other data items are functionally dependent on the same part of the key, place them in the new table also
3. Make the partial primary key copied from the original table the primary key for the new table. Place all items that appear in the repeating group in a new table

**Example 1 (Not 2NF)**

Scheme □ {Title, PubId, AuId, Price, AuAddress}

1. Key □ {Title, PubId, AuId}
2. {Title, PubId, AuID} □ {Price}
3. {AuID} □ {AuAddress}
4. AuAddress does not belong to a key

5. AuAddress functionally depends on AuId which is a subset of a key

**Example 1 (Convert to 2NF)**

Old Scheme □ {Title, PubId, AuId, Price, AuAddress}

New Scheme □ {Title, PubId, AuId, Price}

New Scheme □ {AuId, AuAddress}

**3NF:**

This form dictates that all **non-key** attributes of a table must be functionally dependent on a candidate key i.e. there can be no interdependencies among non-key attributes.

For a table to be in 3NF, there are two requirements

- The table should be second normal form
- No attribute is transitively dependent on the primary key

**Example (Not in 3NF)**

Scheme □ {Title, PubID, PageCount, Price }

1. Key □ {Title, PubId}
2. {Title, PubId} □ {PageCount}
3. {PageCount} □ {Price}
4. Both Price and PageCount depend on a key hence 2NF
5. Transitively {Title, PubID} □ {Price} hence not in 3NF

**Example 1 (Convert to 3NF)**

Old Scheme □ {Title, PubID, PageCount, Price }

New Scheme □ {PubID, PageCount, Price}

New Scheme □ {Title, PubID, PageCount}

**b. Outline constraints in SQL.**

**(6 Marks)**

**Answer:**

## SQL Create Constraints

Constraints can be specified when the table is created with the `CREATE TABLE` statement, or after the table is created with the `ALTER TABLE` statement.

## Syntax

```
CREATE TABLE table_name (

  column1 datatype constraint,

  column2 datatype constraint,

  column3 datatype constraint,

  );
```

# SQL Constraints

SQL constraints are used to specify rules for the data in a table.Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified

**c. Identify the given Relation R(ABCDE) and its instance, check whether FDS given hold or not. Give reasons.**

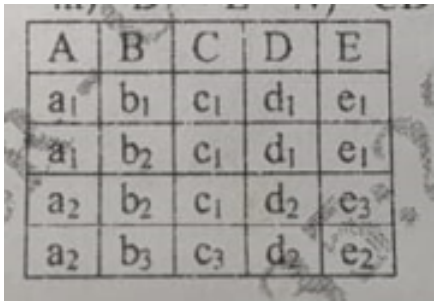
**FDS:**

**i) A → B**

**ii) B → C**

**iii) D → E**

**iv) CD → E**



| A | B | C | D | E |
|---|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ | $e_1$ |
| $a_1$ | $b_2$ | $c_1$ | $d_1$ | $e_1$ |
| $a_2$ | $b_2$ | $c_1$ | $d_2$ | $e_3$ |
| $a_2$ | $b_3$ | $c_3$ | $d_2$ | $e_2$ |

**Answer:**

| FD | Holds? | Reason |
|---|---|---|
| A → B | ❌ | A = $a_1$ maps to $b_1$ and $b_2$ (not unique) |
| B → C | ✅ | Each B value maps to exactly one C value |
| D → E | ❌ | D = $d_2$ maps to $e_2$ and $e_3$ (not unique) |
| CD → E | ✅ | Each (C,D) pair maps to exactly one E value |

**Proof:**

In a relation **R**, a **functional dependency X → Y** holds **if and only if**:

> For any two tuples $t_1$ and $t_2$ in relation R,
> **if** $t_1[X] = t_2[X]$,
> **then** $t_1[Y] = t_2[Y]$.

In simpler terms:

- If two rows (tuples) have **equal values for the attributes on the left side (X)**,
- Then they **must also have equal values for the attributes on the right side (Y)**.

**Q.6**

**a. What is Multivalued dependency? Explain 4NF and 5NF with suitable example.**

**(6 Marks)**

**Answer:**

**Multivalued dependency:**

A table is said to have multi-valued dependency, if the following conditions are true,

- For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
- Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
- And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency

**4NF:**

A relation schema R is in 4NF with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:

– $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
– $\alpha$ is a super key for schema R

• If a relation is in 4NF it is in BCNF

**Example (Not in 4NF)**

Scheme □ {MovieName, ScreeningCity, Genre)
Primary Key: {MovieName, ScreeningCity, Genre)

1. All columns are a part of the only candidate key, hence BCNF
2. Many Movies can have the same Genre
3. Many Cities can have the same movie
4. Violates 4NF

| Movie | ScreeningCity | Genre |
|---|---|---|
| Hard Code | Los Angles | Comedy |
| Hard Code | New York | Comedy |
| Bill Durham | Santa Cruz | Drama |
| Bill Durham | Durham | Drama |
| The Code Warrier | New York | Horror |

1. the table should not have any Multi-valued Dependency (or), if so, Move the two multi-valued relations to separate tables
2. It should be in the **Boyce-Codd Normal Form**
3. Identify a primary key for each of the new entity.

**Example 1 (Convert to 4NF)**

Old Scheme □ {MovieName, ScreeningCity, Genre}

New Scheme □ {MovieName, ScreeningCity}

New Scheme □ {MovieName, Genre}

| Movie | Genre |
|---|---|
| Hard Code | Comedy |
| Bill Durham | Drama |
| The Code Warrier | Horror |

| Movie | ScreeningCity |
|---|---|
| Hard Code | Los Angles |
| Hard Code | New York |
| Bill Durham | Santa Cruz |
| Bill Durham | Durham |
| The Code Warrier | New York |

**5NF:**

- Join dependencies constrain the set of legal relations over a schema R to those relations for which a given decomposition is a lossless-join decomposition.
- Let R be a relation schema and R1, R2, ...,Rn be a decomposition of R. If R = R1 ∪ R2 ∪ ... ∪ Rn, we say that a relation r(R) satisfies the join dependency *(R1, R2, ...,Rn) if:

$$r = \Pi_{R_1}(r) \quad \Pi_{R_2}(r) \quad \ldots\ldots \quad \Pi_{R_n}(r)$$

A join dependency is trivial if one of the $R_i$ is R itself.

- A join dependency *(R1, R2) is equivalent to the multivalued dependency R1 ∩ R2 →→ R2. Conversely, α →→ β *is* equivalent

to *(α ∪ ( R - β ), α ∪ β)

- However, there are join dependencies that are not equivalent to any multivalued dependency.

| Course | Instructor | TextBook_Author |
|---|---|---|
| Management | X | Churchil |
| Management | Y | Peters |
| Management | Z | Peters |
| Finance | A | Weston |
| Finance | A | Gilbert |

**Course**

| Course | Instructor |
|---|---|
| Management | X |
| Management | Y |
| Management | Z |
| Finance | A |

**Textbook_Author**

| Course | TextBook_Author |
|---|---|
| Management | Churchil |
| Management | Peters |
| Finance | Weston |
| Finance | Gilbert |

**b. Outline the informal design guidelines for relational schema.**

**(6 Marks)**

**Answer:**

Four informal guidelines that may be used as measures to determine the quality of relation schema design:

1. Making sure that the semantics of the attributes is clear in the schema
2. Reducing the redundant information in tuples
3. Reducing the NULL values in tuples
4. Disallowing the possibility of generating spurious tuples

These measures are not always independent of one another

**c. Consider relation R with the following functional dependency:**
**EMPPROJ (SSN, Pnumber, Hours, Ename, Pname, Plocation)**

**SSN, Pnumber → Hours**

**SSN → Ename**

**Pnumber → Pname, Plocation**

**Is it 2NF? Verify? If no, give reason.**

**(8 Marks)**

**Answer:**

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

1. It is in 1NF (no multivalued attributes), AND
2. No partial dependency exists, i.e., no non-prime attribute is functionally dependent on a part of a candidate key.

Candidate Key = {SSN, Pnumber}

Prime attributes = SSN, Pnumber

Non-prime attributes = Hours, Ename, Pname, Plocation

1.SSN, Pnumber → Hours

- Full key → attribute ⇒ **Allowed**

2.SSN → Ename

SSN is **part of candidate key**

Ename is a **non-prime attribute**

⇒ **Partial Dependency ⇒ Violates 2NF**

3.Pnumber → Pname, Plocation

Pnumber is **part of candidate key**

Pname, Plocation are **non-prime attributes**

⇒ **Partial Dependency ⇒ Violates 2NF**

The relation is *not in 2NF*.There are **partial dependencies**:

- **SSN → Ename**

- **Pnumber → Pname, Plocation**

These cause **non-prime attributes** to be dependent on **part of a candidate key**, which violates 2NF rules.

**Q.7**

**a. Consider the following schema for a company database:**
**Employee (FName, LName, SSN, Address, Sex, Salary, Dno, Super_SSN)**
**Department (Dname, Dnumber, Mgr_SSN, Mgr_start_date)**
**Project (Pname, Pnumber, Plocation, Dnum)**
**WORKS_on (Essn, Pno, Hours)**
**DEPENDENT (Essn, Dependent_name, Sex, Bdate, Relationship)**

**Write the SQL queries for the following:**
**i) List the names of managers who have at least one dependent (use correlated nested).**
**ii) Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.**
**iii) For each project, retrieve the project number, project name, and the number of employees who work on that project.**
**iv) Retrieve the SSN of all employees who work on project number 1, 2, or 3 (Use IN).**
**v) Find the sum of the salaries of all employees of the 'Research' department as well as maximum salary, minimum salary, average salary in this department.**

**Answer:**

**I. List the names of managers who have at least one dependent (use correlated nested)**

```
SELECT E.FName, E.LName FROM Employee E

WHERE E.SSN IN (

   SELECT Mgr_SSN

   FROM Department D

   WHERE EXISTS (

      SELECT 1

      FROM Dependent Dep

      WHERE Dep.Essn = D.Mgr_SSN

   ));
```

**ii) Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.**

```
SELECT E.FName, E.LName

FROM Employee E

JOIN Dependent D ON E.SSN = D.Essn

WHERE E.FName = D.Dependent_name AND E.Sex = D.Sex;
```

**iii) For each project, retrieve the project number, project name, and the number of employees who work on that project.**

```
SELECT P.Pnumber, P.Pname, COUNT(W.Essn) AS Num_Employees FROM Project P

LEFT JOIN WORKS_on W ON P.Pnumber = W.Pno

GROUP BY P.Pnumber, P.Pname;
```

**iv) Retrieve the SSN of all employees who work on project number 1, 2, or 3 (Use IN).**

```
SELECT DISTINCT Essn

FROM WORKS_on WHERE Pno IN (1, 2, 3);
```

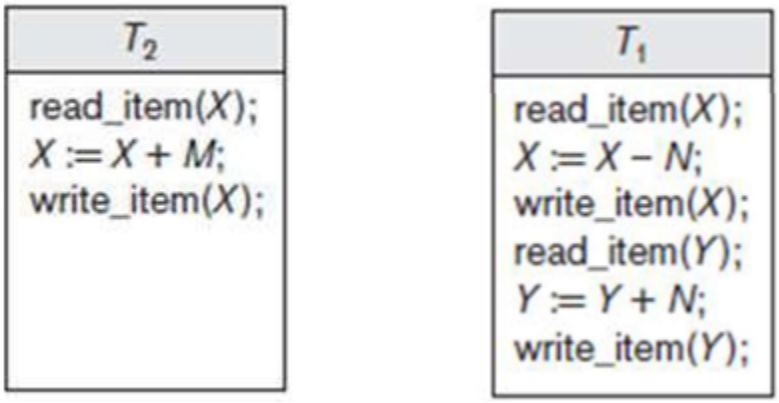
**v) Find the sum of the salaries of all employees of the 'Research' department as well as** maximum salary, minimum salary, average salary in this department.

```
SELECT

   SUM(E.Salary) AS Total_Salary,

   MAX(E.Salary) AS Max_Salary,

   MIN(E.Salary) AS Min_Salary,

   AVG(E.Salary) AS Avg_Salary

FROM Employee E

JOIN Department D ON E.Dno = D.Dnumber

WHERE D.Dname = 'Research';
```

**7b.Why concurrency control is needed? Demondtrate with an example? 10 Marks**

**Answer:**

- Several problems can occur when concurrent transactions execute in an uncontrolled manner
- Example:
  - We consider an Airline reservation DB
  - Each records is stored for an airline flight which includes Number of reserved seats among other information.
  - Types of problems we may encounter:
    1. The Lost Update Problem
    2. The Temporary Update (or Dirty Read) Problem
    3. The Incorrect Summary Problem
    4. The Unrepeatable Read Problem




- Transaction T1
  - transfers N reservations from one flight whose number of reserved seats is stored in the database item named X to another flight whose number of reserved seats is stored in the database item named Y.
- Transaction T2
  - reserves M seats on the first flight (X)

**8a.Consider the following schedule.The actions are listed in the order they are scheduled and prefixed with the transaction name.**

**S1:T1:R(X),T2:R(X),T1:W(Y),T2:W(Y),T1:R(Y),T2:R(Y)**

**S2:T3:W(X),T1:R(X),T1:W(Y),T2:R(Z),T2:W(Z),T3:R(Z)**

**For each schedule answer the following:**

**i)What is the schedule precedence graph for the schedule?**

**ii)Is the schedule conflict serializable?If so what are all the conflicts equivalent serial schedules?**

**iii)Is the schedule view serializable?If so what are all the view equivalent serial schedules? 10 Marks**

**Answer:**

For schedule S1, it is conflict serializable with equivalent serial schedules T1 followed by T2 (T1 -> T2) or T2 followed by T1 (T2 -> T1). For schedule S2, it is not conflict serializable due to the cycle T1 -> T3 -> T2 -> T1. Both schedules are view serializable.

## Schedule S1

### Precedence Graph:

- The precedence graph for S1 has two transactions: T1 and T2.
- There is an edge from T1 to T2 because T1 reads X before T2 reads X (T1 -> T2).
- There is an edge from T1 to T2 because T1 writes Y before T2 reads Y (T1 -> T2).
- There is an edge from T1 to T2 because T1 writes Y before T2 writes Y (T1 -> T2).
- There is an edge from T2 to T1 because T2 reads X before T1 reads X.
- There is an edge from T2 to T1 because T2 writes Y before T1 writes Y.
- There is an edge from T2 to T1 because T2 reads Y before T1 reads Y.
- The precedence graph contains a cycle: T1 -> T2 -> T1.
- T1 -> T2 (and vice versa) is also present due to the conflicting R/W or W/R operations on X and Y between the two transactions.
- Since there's a cycle, S1 is not conflict serializable.
- However, S1 is view serializable. The view equivalent serial schedules are T1 -> T2 and T2 -> T1.

### Conflict Serializability:

- S1 is not conflict serializable because its precedence graph contains a cycle.
- The equivalent serial schedules are T1 -> T2 or T2 -> T1.

### View Serializability:

- S1 is view serializable because it can be transformed into a serial schedule without changing the final state of the data items. The two serial schedules T1 -> T2 and T2 -> T1 would result in the same final values for X and Y.

## Schedule S2

### Precedence Graph:

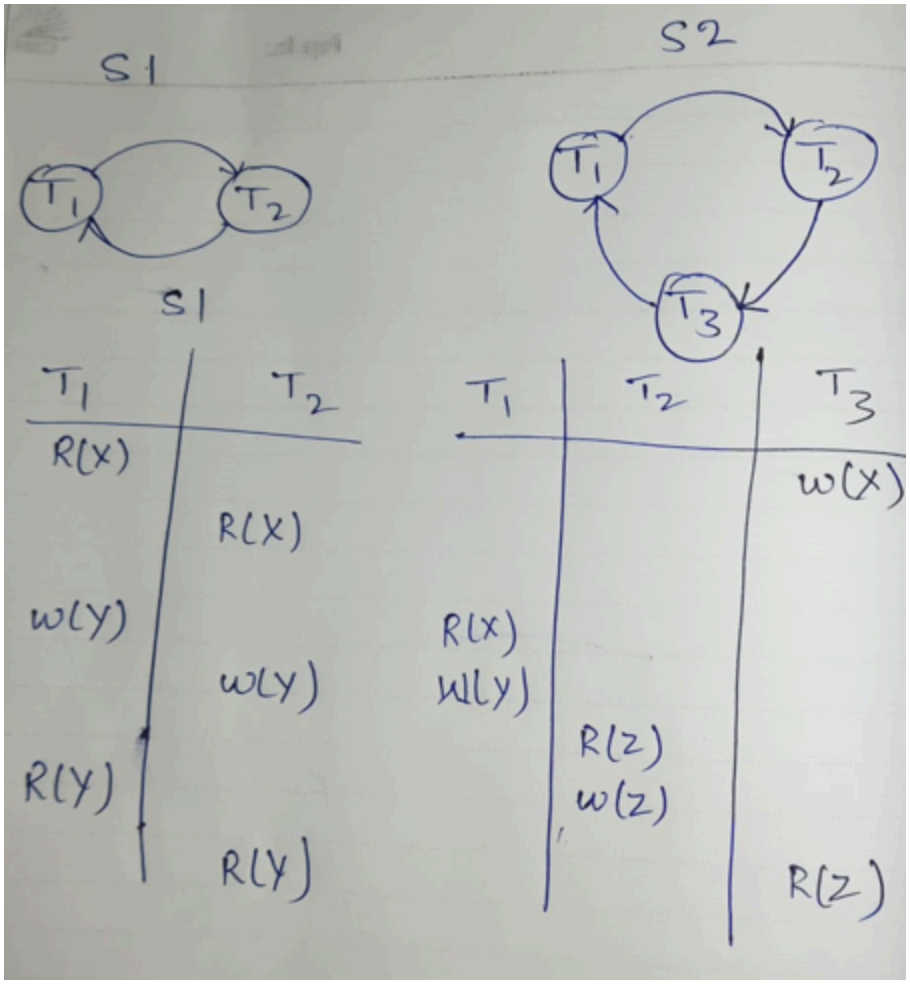
- The precedence graph for S2 has three transactions: T1, T2 and T3.
- There is an edge from T3 to T1 because T3 writes X before T1 reads X (T3 -> T1).
- There is an edge from T1 to T2 because T1 writes Y before T2 reads Z (T1 -> T2).
- There is an edge from T2 to T3 because T2 writes Z before T3 reads Z (T2 -> T3).
- There is a cycle: T1 -> T2 -> T3 -> T1.
- Since there's a cycle, S2 is not conflict serializable.

### Conflict Serializability:

- S2 is not conflict serializable because its precedence graph contains a cycle.

### View Serializability:

- S2 is view serializable because it can be transformed into a serial schedule. The view equivalent serial schedules can be any permutation of T1, T2, and T3: T1 -> T2 -> T3, T1 -> T3 -> T2, T2 -> T1 -> T3, T2 -> T3 -> T1, T3 -> T1 -> T2, or T3 -> T2 -> T1.




**b.Explain triggers with example write a trigger in SQL to call a procedure "Inform_Supervisor" whenever an employees salary is greater than the salary if his or her direct supervisor in the COMPANY database. 10Marks**

**Answer:**

Whenever an employee's salary is greater than their supervisor's salary, a procedure `Inform_Supervisor` must be called.

**Employee** Table:

Employee(FName, LName, SSN, Address, Sex, Salary, Dno, Super_SSN)

SSN: Employee ID

`Super_SSN`: Supervisor's SSN

`Salary`: Employee's salary

**Procedure**:

CREATE PROCEDURE Inform_Supervisor(empSSN CHAR(9))

BEGIN

END;

# Trigger Logic

- On UPDATE of an employee's salary
- If `NEW.Salary > Supervisor's Salary`, then call `Inform_Supervisor(NEW.SSN)`

```
DELIMITER //

CREATE TRIGGER Check_Salary_Increase

AFTER UPDATE ON Employee

FOR EACH ROW

BEGIN

    DECLARE super_salary DECIMAL(10,2);

    -- Get supervisor's salary

    SELECT Salary INTO super_salary

    FROM Employee

    WHERE SSN = NEW.Super_SSN;

    -- If employee's salary is now greater than supervisor's,
call procedure

    IF NEW.Salary > super_salary THEN

        CALL Inform_Supervisor(NEW.SSN);

    END IF;

END;
```

```
//

DELIMITER ;
```

## Q.9 a. Describe the two - phase locking protocol for concurrency control provide example to illustrate how it ensures serializability in transaction schedule

The Two-Phase Locking (2PL) Protocol is a key technique used in DBMS to manage how multiple concurrent transactions access and modify data, with two phases: growing (where all locks are acquired) and shrinking (where locks are released)

**Two Phases:**

1. **Growing Phase (Expanding Phase):**
    - Transaction can acquire locks (shared or exclusive)
    - Cannot release any locks
    - Transaction grows by acquiring more locks
2. **Shrinking Phase (Contracting Phase):**
    - Transaction can release locks
    - Cannot acquire new locks
    - Transaction shrinks by releasing locks

**How 2PL Ensures Serializability:**

The Two-Phase Locking Protocol ensures serializability by controlling when locks can be acquired and released, ensuring that while one transaction is accessing a data item, other transactions cannot modify that particular data item VaiaExploredatabase.

**Example:** Consider two transactions T1 and T2:

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

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

**With 2PL:**

T1: Lock-S(A), Read(A), Lock-X(A), Write(A), Lock-S(B), Read(B),

Lock-X(B), Write(B), Unlock(A), Unlock(B)

T2: Must wait for T1 to release locks before proceeding

This ensures conflict serializability by preventing concurrent access to shared data items during critical operations.

## b. Explain the characteristics of NOSQL system

Key Characteristics:

1. Schema Flexibility: NoSQL databases don't require a fixed schema, allowing dynamic addition of fields
2. Horizontal Scalability: NoSQL databases like MongoDB are known for flexibility, scalability, and high performance, widely used by companies like Adobe, Uber, IBM, and Google for big data applications MongoDB CRUD Operations - GeeksforGeeks
3. High Performance: Optimized for specific data models and access patterns
4. Distributed Architecture: Built for distributed computing environments
5. CAP Theorem Trade-offs: Choose between Consistency, Availability, and Partition tolerance
6. Types: Document stores, Key-value stores, Column-family, Graph databases
7. BASE Properties: Basically Available, Soft state, Eventually consistent (vs ACID)

**OR**

## Q.10 a. Explain binary locks and shared lock with algorithm

**Binary Locks:**

- Simple locking mechanism with two states: LOCKED (1) or UNLOCKED (0)
- Only one transaction can hold the lock at a time

Algorithm for Binary Locks:

**LOCK(X):**

**While Lock(X) = 1 do**

**Wait**

**Lock(X) = 1**

**UNLOCK(X):**

**Lock(X) = 0**

**Shared Locks:**

- Multiple transactions can read simultaneously
- Exclusive access for write operations

**Types:**

1. Shared Lock (S-lock): For read operations
2. Exclusive Lock (X-lock): For write operations

**Algorithm for Shared/Exclusive Locks:**

**LOCK-S(X):**

```
If Lock(X) = "unlocked" then
    Lock(X) = "read-locked"
    no_of_reads(X) = 1
Else if Lock(X) = "read-locked" then
    no_of_reads(X) = no_of_reads(X) + 1
Else wait
```

**LOCK-X(X):**

```
If Lock(X) = "unlocked" then
    Lock(X) = "write-locked"
Else wait
```

**UNLOCK(X):**

```
If Lock(X) = "write-locked" then
    Lock(X) = "unlocked"
Else if Lock(X) = "read-locked" then
    no_of_reads(X) = no_of_reads(X) - 1
    If no_of_reads(X) = 0 then
        Lock(X) = "unlocked"
```

## b. Explain MongoDB data model, CRUD operations and distributed system characteristics

### Characteristics

**MongoDB Data Model: MongoDB stores data in flexible, JSON-like documents rather than traditional relational tables.**

- **Document-based: Data stored in BSON (Binary JSON) format**
- **Collections: Groups of documents (similar to tables)**
- **Flexible Schema: Documents in same collection can have different structures**
- **Embedded Documents: Support for nested data structures**
- **Arrays: Native support for array data types**

**CRUD Operations:**

**MongoDB supports four primary CRUD operations: Create (using insertOne() or insertMany()), Read, Update, and Delete operations.**

**Create:**

```
javascript
db.collection.insertOne({name: "John", age: 30})
```

1. `db.collection.insertMany([{...}, {...}])`

**Read:**

```
javascript
db.collection.find({name: "John"})
```

2. db.collection.findOne({_id: ObjectId(...)})

**Update:**

```
javascript
db.collection.updateOne({name: "John"}, {$set: {age: 31}})
```

3. db.collection.updateMany({}, {$inc: {age: 1}})

**Delete:**

```
javascript
db.collection.deleteOne({name: "John"})
```

4. db.collection.deleteMany({age: {$lt: 18}})

**Distributed System Characteristics:**

1. **Horizontal Scaling (Sharding): Distributes data across multiple servers**
2. **Replication: Master-slave configuration for high availability**
3. **Auto-failover: Automatic switching to backup servers**
4. **Load Balancing: Distributes queries across replica sets**
5. **Consistency Models: Configurable read/write concerns**
6. **Partition Tolerance: Continues operation despite network failures**
7. **Geographic Distribution: Support for multi-region deployments**