Solution for VTU Question Paper- JUNE/JULY-2023

Database Management System (22MCA21)

Q. No. 1a)Explain the database system environment with neat diagram

1b) Discuss the characteristics and advantages of Database Approaches

Describe the characteristics ofDatabase Approach

1)Self-Describing Nature of a Database System.
2)Isolation between Programs and Data, and Data Abstraction
3)Support for Multiple Views of the Data
4)Sharing of knowledge and Multi-user Transaction Processing

Advantages of Database Approach:

The following are the advantages of using a DBMS:

ADVANTAGES OF USING THE DBMS APPROACH

The following are the advantages of using a DBMS:

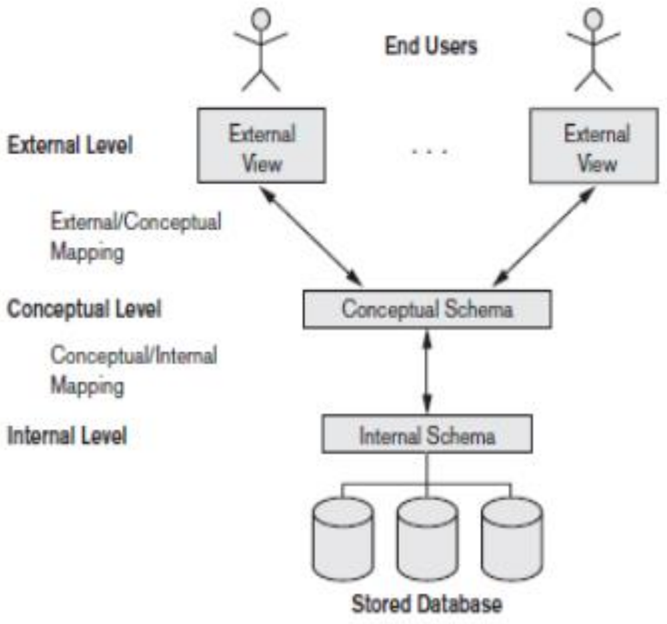
- Controlling Redundancy.
- Restricting Unauthorized Access.
- Providing Persistent Storage for Program Objects.
- Providing Storage Structures and Search Techniques for Efficient Query Processing.
- Providing Backup and Recovery.
- Providing Multiple User Interfaces.
- Representing Complex Relationships among Data.
- Enforcing Integrity Constraints.
- Permitting Inferencing and Actions Using Rules.

Q. No. (2)a. Explain with proper diagram, the 3-schema architecture of DBMS.

<u>The Three-Schema Architecture</u>

- The three schema architecture was proposed to help in achieving and visualizing the characteristics of the database approach. The below figure illustrates the Three Schema Architecture.

Figure 2.2
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:
  - 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.
  - 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.
  - 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.

- The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system. The processes of transforming requests and results between levels are called **mappings**.

Data Independence

- **Data independence** is the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.
- The following are the types of data independence:
  - **Logical data independence**
  - **Physical data independence**

(2)b. What are the different types of attributes? Explain with example.

- Types of attributes
  - simple attributes.
  - Composite attributes.
  - Singlevalued attributes.
  - Multivalued attributes.
  - stored and derived attributes.
- Simple attributes:
  - Attributes that are not divisible are called **simple** or **atomic attributes**.
  - Example: SSN and gender attribute of an employee entity.
- Composite attributes:
  - Attributes that can be divided into smaller subparts, which represent more basic attributes with independent meanings are called as **Composite attributes**.
  - For Example, the Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State, and Zip.

- Composite attributes can form a hierarchy like Number, Street, and Apartment_number, as shown in below Figure 7.4.
- Composite attributes are useful to model situations in which a user sometimes refers to the composite attribute as a unit but at other times refers specifically to its components.

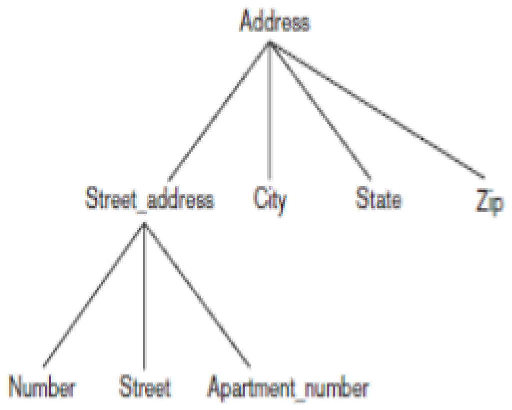


Figure 7.4
A hierarchy of composite attributes.

- Single-Valued attributes**:**
    - The attributes that have a single value for a particular entity are called as **single-valued attributes**.
    - Example: Age and SSN are single-valued attributes of an employee.
- Multivalued attributes:
    - The attribute that can have a set of values for the same entity is called **multivalued attribute**.
    - Examples: Colors attribute for a car or a College_degrees attribute for a person.
    - A multivalued attribute may have lower and upper bounds to constrain the *number of values* allowed for each individual entity.
    - For example, the Colors attribute of a car may be restricted to have between one and three values, if we assume that a car can have three colors at most.

colors at most.

- Stored versus Derived Attributes:

  In some cases, two (or more) attribute values are related such as the Age and Birth_date attributes of a person. For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's Birth_date. The Age attribute is hence called a **derived attribute** and is said to be **derivable from** the Birth_date attribute, which is called a **stored attribute.**

- Complex Attributes:
  - Composite and Multivalued attributes can be nested arbitrarily.Such attributes are called as complex attributes.
  - Composite attributes can be represented by grouping components of a composite attribute between parentheses () and separating the components with commas, and by displaying multivalued attributes between braces { }.

- For example, if a person can have more than one residence and each residence can have a single address and multiplephones, an attribute Address_phone for a person can be specified as shown in below Figure 7.5. Both Phone and Address are themselves composite attributes.

{Address_phone({Phone(Area_code,Phone_number)},Address(Street_address (Number,Street,Apartment_number),City,State,Zip))}

Figure 7.5
A complex attribute: Address_phone.

- <u>NULL Values</u>
    - In some cases, a particular entity may not have an applicable value for an attribute. For example, a College_degrees attribute of a person entity applies only to people with college degrees. For such situations, a special value called NULL is created. a person with no college degree would have NULL for College_degrees.here meaning of NULL is not applicable.
    - NULL can also be used if we do not know the value of an attribute. For a particular entity such as employee, if we do not know the home phone number of an employee we can use NULL value. The meaning of NULL here is **unknown**.
    - The **unknown** category of NULL can be further classified into two cases. The first case arises when it is known that the attribute value exists but is **missing**. For instance, if the Height attribute of a person is listed as NULL. The second case arises when it is not known whether the attribute value exists, for example, if the Home_phone attribute of a person is NULL.

Q. No. (3)a. Explain unary Operation Select and prove it is commutative.

The SELECT Operation

- ✓ The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a¬ selection condition.
- ✓ It restricts the tuples in a relation to only those tuples that¬ satisfy the condition.
- ✓ It can also be visualized as a horizontal partition of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.
- ✓ For example, to select the EMPLOYEE tuples whose department is 4, or those whose salary is greater than $30,000

$$\boxed{\begin{array}{c}\sigma Dno=4(EMPLOYEE) \\ \sigma Salary>30000(EMPLOYEE)\end{array}}$$

- ✓ In general, the SELECT operation is denoted by

  σ<selection condition>(R)

  where the symbol σ (sigma) is used to denote the SELECT operator and the selection condition is a Boolean expression (condition) specified on the attributes of relation R.
- ✓ The Boolean expression specified in is made up of a number of clauses of the form :

  <attribute name><comparison op><constant value>

  Or

  <attribute name><comparison op><attribute name>
- ✓ Clauses can be connected by the standard Boolean operators and, or, and not to form a general selection condition.

- ✓ The Boolean conditions AND, OR, and NOT have their normal interpretation, as follows:
  - ■ (cond1 AND cond2) is TRUE if both (cond1) and (cond2) are TRUE; otherwise,it is FALSE.
  - ■ (cond1 OR cond2) is TRUE if either (cond1) or (cond2) or both are TRUE; otherwise, it is FALSE.
  - ■ (NOT cond) is TRUE if cond is FALSE; otherwise, it is FALSE.
- ✓ The SELECT operator is unary; that is, it is applied to a single relation. Hence, selection conditions cannot involve more than one tuple.
- ✓ The degree of the relation resulting from a SELECT operation—its number of attributes—is the¬ same as the degree of R.
- ✓ The SELECT operation is commutative; that is,

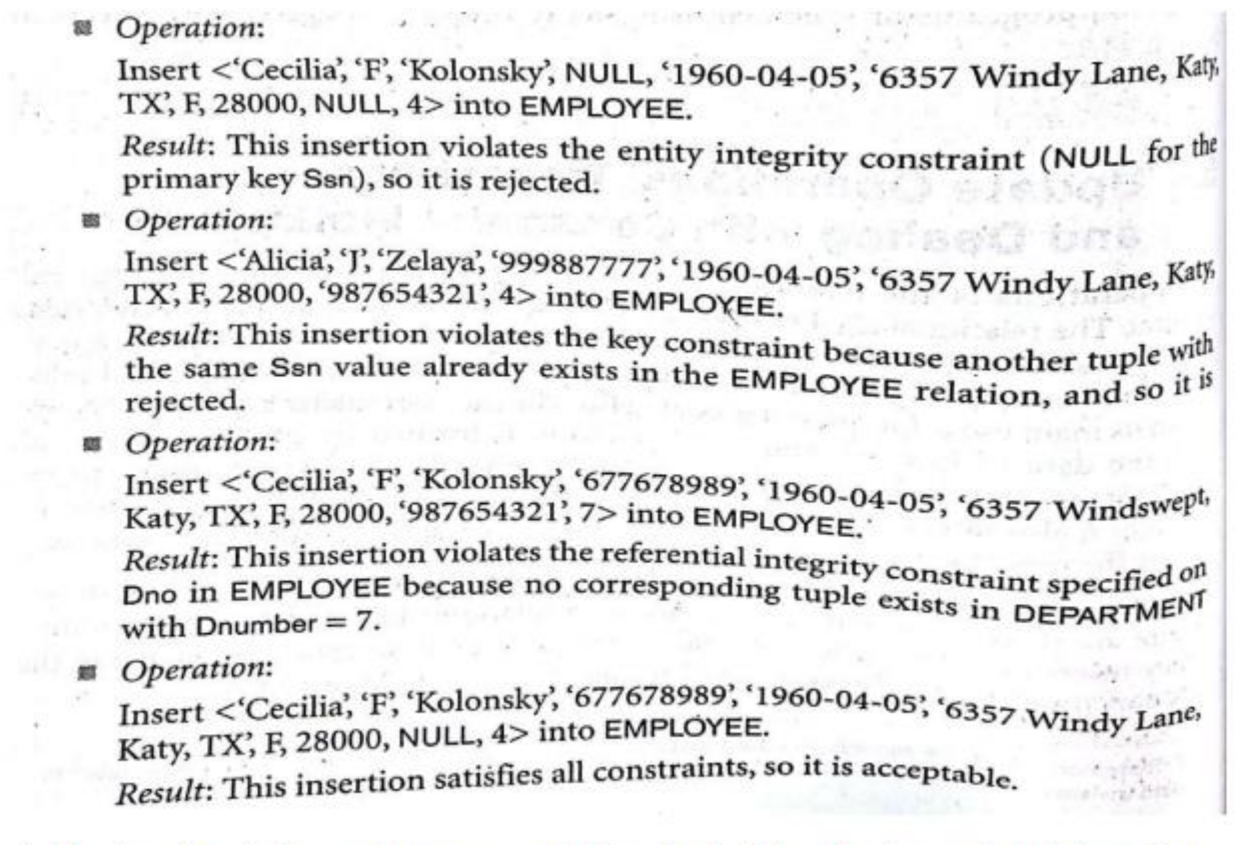
$$\sigma\ (\text{cond1})(\sigma(\text{cond2})(R)) = \sigma(\text{cond2})(\sigma(\text{cond1})(R))$$

(3)b.Explain schema update operations, with a suitable examples.

- ✓ There are three basic operations that can change the states of relations in the database: Insert, Delete, and Update (or Modify).
- ✓ Insert is used to insert one or more new tuples in a relation.
- ✓ Delete is used to delete tuples.
- ✓ Update (or Modify) is used to change the values of some attributes in existing tuples.

The Insert Operation:

- ✓ The Insert operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R.
- ✓ Insert can violate any of the four types of constraints.
  1. **Domain constraints** can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type.
  2. **Key constraints** can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R).
  3. **Entity integrity** can be violated if any part of the primary key of the new tuple t is NULL.
  4. **Referential integrity** can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.



- *Operation*:

  Insert <'Cecilia', 'F', 'Kolonsky', NULL, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, NULL, 4> into EMPLOYEE.

  Result: This insertion violates the entity integrity constraint (NULL for the primary key Ssn), so it is rejected.
- *Operation*:

  Insert <'Alicia', 'J', 'Zelaya', '999887777', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, '987654321', 4> into EMPLOYEE.

  Result: This insertion violates the key constraint because another tuple with the same Ssn value already exists in the EMPLOYEE relation, and so it is rejected.
- *Operation*:

  Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windswept, Katy, TX', F, 28000, '987654321', 7> into EMPLOYEE.

  Result: This insertion violates the referential integrity constraint specified on Dno in EMPLOYEE because no corresponding tuple exists in DEPARTMENT with Dnumber = 7.
- *Operation*:

  Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, NULL, 4> into EMPLOYEE.

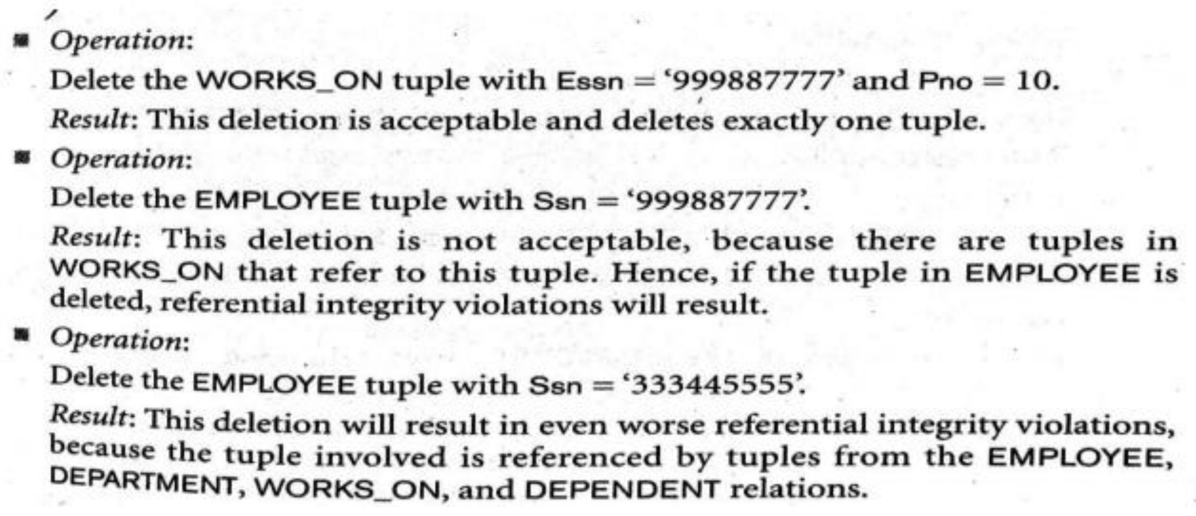
  Result: This insertion satisfies all constraints, so it is acceptable.

- ✓ If an insertion violates one or more constraints, the default option is to reject the insertion.
- ✓ Another option is to attempt to correct the reason for rejecting the insertion, but this is typically not used for violations caused by Insert; rather, it is used more often in correcting violations for Delete and Update.

<u>The Delete Operation</u>

- ✓ The Delete operation can violate only referential integrity.

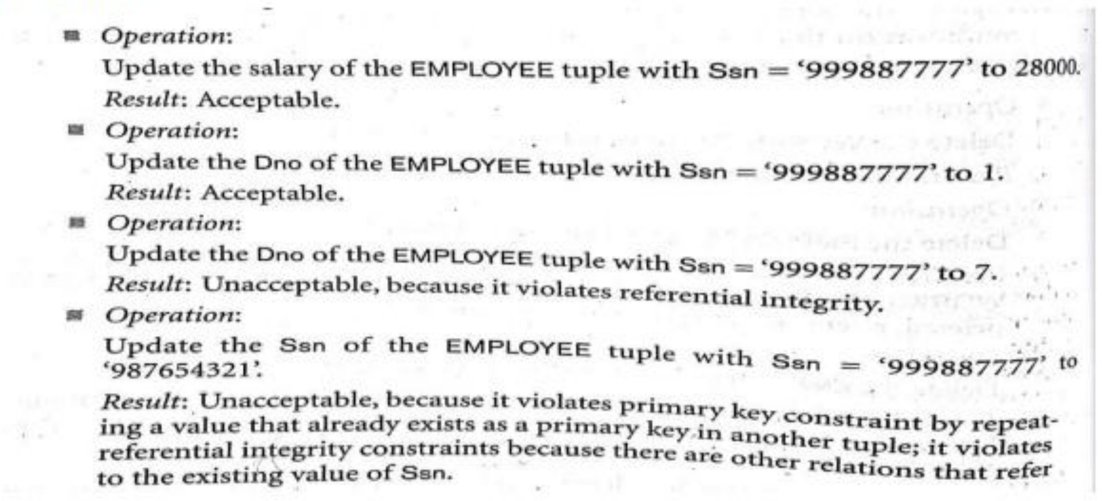
- ✓ This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database.
- ✓ To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted.



- *Operation*:
  Delete the WORKS_ON tuple with Essn = '999887777' and Pno = 10.
  Result: This deletion is acceptable and deletes exactly one tuple.
- *Operation*:
  Delete the EMPLOYEE tuple with Ssn = '999887777'.
  Result: This deletion is not acceptable, because there are tuples in WORKS_ON that refer to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result.
- *Operation*:
  Delete the EMPLOYEE tuple with Ssn = '333445555'.
  Result: This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.

The Update Operation

The Update (or Modify) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified.



- *Operation*:
  Update the salary of the EMPLOYEE tuple with Ssn = '999887777' to 28000.
  Result: Acceptable.
- *Operation*:
  Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 1.
  Result: Acceptable.
- *Operation*:
  Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 7.
  Result: Unacceptable, because it violates referential integrity.
- *Operation*:
  Update the Ssn of the EMPLOYEE tuple with Ssn = '999887777' to '987654321'.
  Result: Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of Ssn.

4.a. with a suitable example, explain join and division operation in relational algebra.

The DIVISION Operation

- ✓ The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimesoccurs in database applications.
- ✓ example is Retrieve the names of employees who work on all the projects that 'John Smith' works on. To express this query using the DIVISION operation, proceed as follows. First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH_PNOS:

  SMITH ← σFname='John' AND Lname='Smith'(EMPLOYEE)

  SMITH_PNOS ← πPno(WORKS_ON $\bowtie$ Essn=SsnSMITH)
- ✓ Next, create a relation that includes a tuple whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

  SSN_PNOS ← πEssn, Pno(WORKS_ON)
- ✓ Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:

  SSNS(Ssn) ← SSN_PNOS ÷ SMITH_PNOS

  RESULT ← πFname, Lname(SSNS * EMPLOYEE)
- ✓ In general, the DIVISION operation is applied to two relations R(Z) ÷ S(X), where the attributes of R are a subset of the attributes of S; that is, $X \subseteq Z$. Let Y be the set of attributes of R that are not attributes of S.
- ✓ The DIVISION operation is defined for convenience for dealing with queries that involve universal quantification or the all condition.

- ✓ The main difference between CARTESIAN PRODUCT and JOIN are ,In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result.
- ✓ The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples. Each tuple combination for which the join condition evaluates to TRUE is included in the resulting relation Q as a single combined tuple.

  - ✓ A general join condition is of the form
    <condition>AND<condition> AND … AND<condition>
    where each <condition> is of the form Ai θ Bj , 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 {=, <,>,< , ≥, ≠}.
    A JOIN operation with such a general join condition is called a THETA JOIN.

The JOIN Operation

- ✓ The JOIN operation, denoted by , is used to combine related tuples from two relations into single "longer" tuples.
- ✓ This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.
- ✓ To illustrate JOIN, suppose that we want to retrieve the name of the manager of each department, as follows:

> DEPT_MGR ← DEPARTMENT $\bowtie$ Mgr_ssn=Ssn EMPLOYEE
> RESULT ← πDname, Lname, Fname(DEPT_MGR)

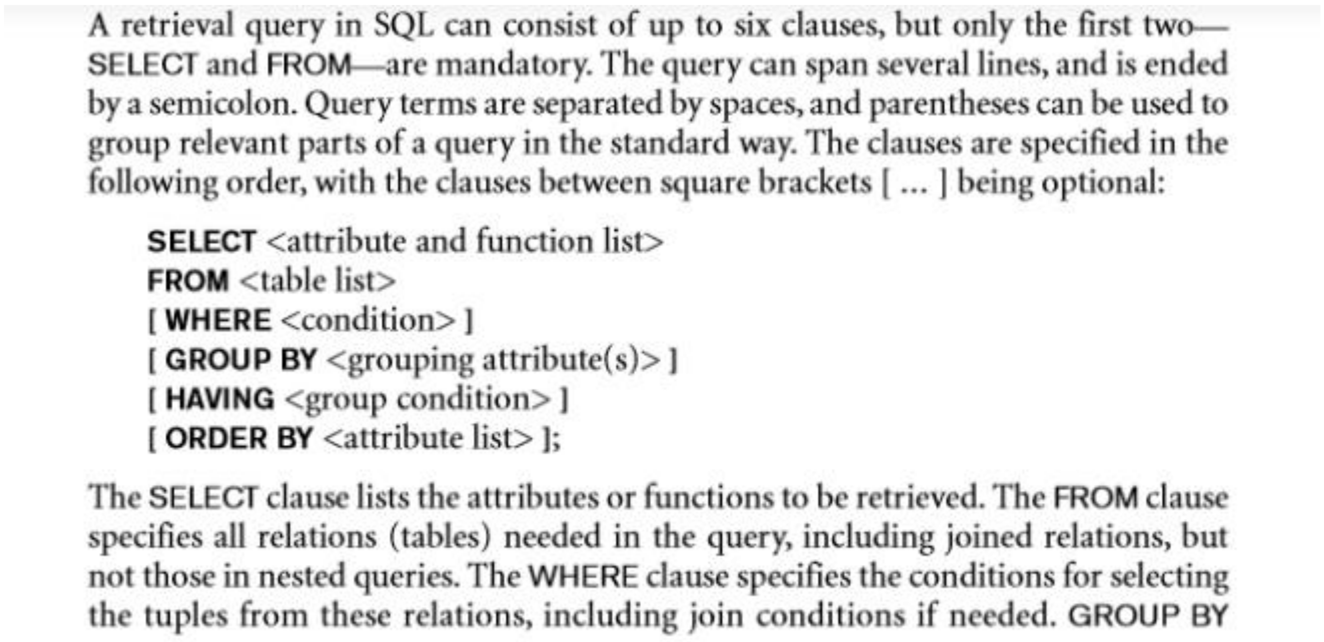
- ✓ The general form of a JOIN operation on two relations R(A1, A2, … , An) and S(B1, B2, … , Bm) is

  R $\bowtie$<join condition>S
- ✓ 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.

4.b. Explain in detail ER-to Relational mapping algorithms.

- **ER-to-Relational Mapping Algorithm**
  - Step 1: Mapping of Regular Entity Types
  - Step 2: Mapping of Weak Entity Types
  - Step 3: Mapping of Binary 1:1 Relation Types
  - Step 4: Mapping of Binary 1:N Relationship Types.
  - Step 5: Mapping of Binary M:N Relationship Types.
  - Step 6: Mapping of Multivalued attributes.
  - Step 7: Mapping of N-ary Relationship Types.

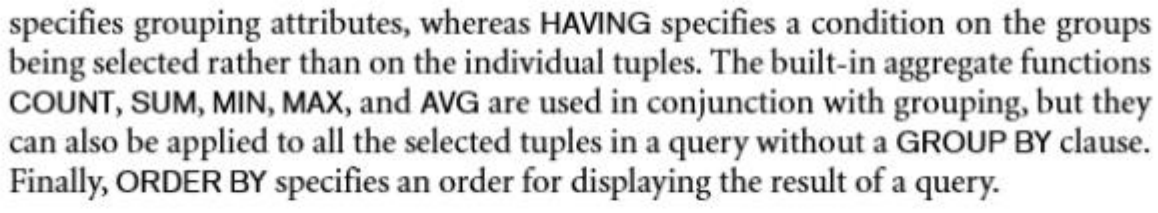
5.a. Expalin with suitable example the basic structure of SQL query.



A retrieval query in SQL can consist of up to six clauses, but only the first two—SELECT and FROM—are mandatory. The query can span several lines, and is ended by a semicolon. Query terms are separated by spaces, and parentheses can be used to group relevant parts of a query in the standard way. The clauses are specified in the following order, with the clauses between square brackets [...] being optional:

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

The SELECT clause lists the attributes or functions to be retrieved. The FROM clause specifies all relations (tables) needed in the query, including joined relations, but not those in nested queries. The WHERE clause specifies the conditions for selecting the tuples from these relations, including join conditions if needed. GROUP BY



specifies grouping attributes, whereas HAVING specifies a condition on the groups being selected rather than on the individual tuples. The built-in aggregate functions COUNT, SUM, MIN, MAX, and AVG are used in conjunction with grouping, but they can also be applied to all the selected tuples in a query without a GROUP BY clause. Finally, ORDER BY specifies an order for displaying the result of a query.

5.b. What are views in SQL? Explain.

1.1 Concept of a View in SQL

- A view is a single table that is derived from one or more base tables or other views
- Views neither exist physically nor contain data itself, it depends on the base tables for its existence
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

1.2 Specification of Views in SQL

Syntax:

```
CREATEVIEWview_nameAS
SELECTcolumn_name(s)
FROMtable_name
WHEREcondition
```

Example

```
CREATE VIEW  WORKS_ON1
AS SELECT    Fname, Lname, Pname, Hours
FROM     EMPLOYEE, PROJECT, WORKS_ON
WHERE   Ssn=Essn  ANDPno=Pnumber ;
```

- We can specify SQL queries on view

Example #

- Retrieve the Last name and First name of all employees who work on 'ProductX'

```
SELECT    Fname, Lname
FROM    WORKS_ON1
WHERE    Pname='ProductX' ;
```

- A view always shows **up-to-date**
- If we **modify** the tuples in the **base tables** on which the view is defined, the view must **automatically reflect** these **changes**
- If we do not need a view any more, we can use the DROP VIEW command

DROP VIEW WORKS_ON1;

OBSERVATIONS ON VIEWS

- ❑ A view with a single defining table is updatable if the view attributes contain the primary key of the base relation, as well as all attributes with the NOT NULL constraint that do not have default values specified
- ❑ Views defined on multiple tables using joins are generally not updatable
- ❑ Views defined using grouping and aggregate functions are not updatable
- ❖ In SQL, the clause WITH HECK OPTION must be added at the end of the view definition if a view is to be updated.

Advantages of Views

- Data independence
- Currency
- Improved security
- Reduced complexity
- Convenience
- Customization
- Data integrity

6.a. In SQL how to handle the Aggregate functions with group by and having clause? With examples.

GROUP BY clause

SQL has a GROUP BY clause . The GROUP BYclause specifies the grouping attributes, which should also appear in the SELECT clause, so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).

In many cases we want to apply the aggregate functions to subgroups of tuples in a relation, where the subgroups are based on some attribute values. For example, we may want to find the average salary of employees in each department or the number of employees who work on each project.In these cases we need to partition the relation into nonoverlapping subsets (or groups) of tuples.Each group (partition) will consist of the tuples that have the same value of some attribute(s), called the grouping attribute(s). We can then apply the function to each such group independently to produce summary information about each group.

Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q24:   SELECT     Dno, COUNT (*), AVG (Salary)
       FROM       EMPLOYEE
       GROUP BY   Dno;
```

In Q24, the EMPLOYEE tuples are partitioned into groups—each group having the same value for the grouping attribute Dno. Hence, each group contains the employees who work in the same department. The COUNT and AVG functions are applied to each such group of tuples. Notice that the SELECT clause includes only the grouping attribute and the aggregate functions to be applied on each group of tuples. Figure 5.1(a) illustrates how grouping works on Q24; it also shows the result of Q24.

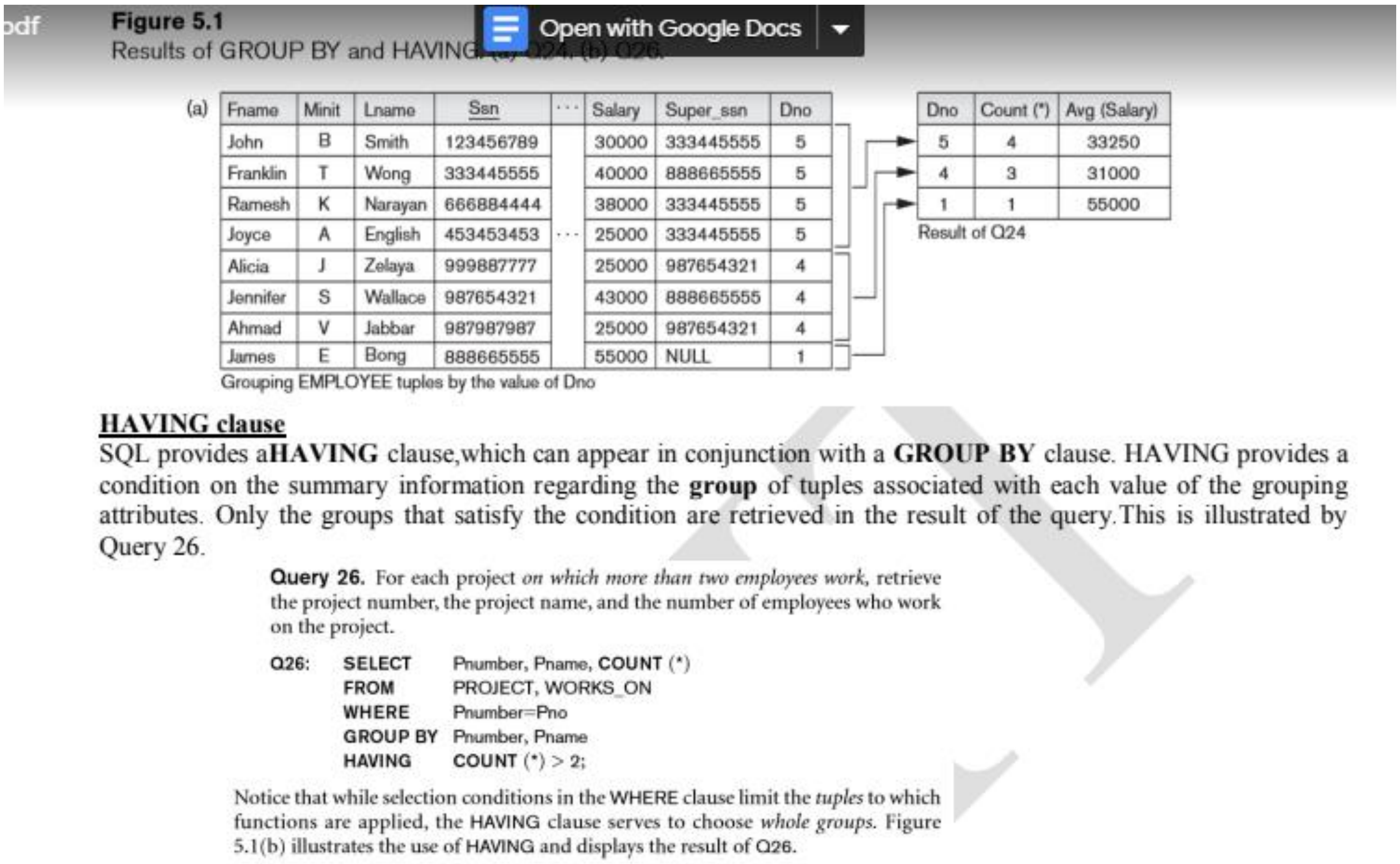


Figure 5.1
Results of GROUP BY and HAVING. (a) Q24. (b) Q26.

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

HAVING clause

SQL provides a**HAVING** clause,which can appear in conjunction with a **GROUP BY** clause. HAVING provides a condition on the summary information regarding the **group** of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result of the query.This is illustrated by Query 26.

Query 26. For each project *on which more than two employees work*, retrieve the project number, the project name, and the number of employees who work on the project.

```
Q26:  SELECT     Pnumber, Pname, COUNT (*)
      FROM       PROJECT, WORKS_ON
      WHERE      Pnumber=Pno
      GROUP BY   Pnumber, Pname
      HAVING     COUNT (*) > 2;
```

Notice that while selection conditions in the WHERE clause limit the *tuples* to which functions are applied, the HAVING clause serves to choose *whole groups*. Figure 5.1(b) illustrates the use of HAVING and displays the result of Q26.

6.b. What are aggregate functions? Explain with an examples.

Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary. **Grouping** is used to create subgroups of tuples before summarization. **Grouping and aggregation** are required in many database applications, and we will introduce their use in SQL through examples.

A number of built-in **aggregate** functions exist: **COUNT, SUM, MAX, MIN, and AVG.**The COUNT function returns the number of tuples or values as specified in aquery.The functions SUM,MAX,MIN,and AVG can be applied to a set or multiset of numeric values and return,respectively,the sum,maximum value,minimum value, and average (mean) of those values.

Query 19. Find the sum of the salaries of all employees,the maximum salary, the minimum salary,and the average salary.

```
SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
FROM EMPLOYEE;
```

If we want to get the preceding function values for employees of a specific department—say, the 'Research' department—we can write Query 20, where the EMPLOYEEtuples are restricted by the WHERE clause to those employees who work for the 'Research'department.

Query 20. Find the sum of the salaries of all employees of the 'Research' department,as well as the maximum salary,the minimum salary,and the average salary in this department.

```
SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
FROM (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
 WHERE Dname='Research';
```

```
Q21:   SELECT   COUNT (*)
       FROM     EMPLOYEE;

Q22:   SELECT   COUNT (*)
       FROM     EMPLOYEE, DEPARTMENT
       WHERE    DNO=DNUMBER AND DNAME='Research';
```

Here the asterisk (*) refers to the *rows* (tuples), so COUNT (*) returns the number of rows in the result of the query. We may also use the COUNT function to count values in a column rather than tuples, as in the next example.

Query 23. Count the number of distinct salary values in the database.

```
Q23:   SELECT   COUNT (DISTINCT Salary)
       FROM     EMPLOYEE;
```

If we write COUNT(SALARY) instead of COUNT(DISTINCT SALARY) in Q23, then duplicate values will not be eliminated. However, any tuples with NULL for SALARY will not be counted. In general, NULL values are discarded when aggregate functions are applied to a particular column (attribute).

6.c. Explain the architecture of JDBC main componenets and types of drivers.

ODBC and JDBC, short for Open DataBase Connectivity and Java DataBase Connectivity, also enable the integration of SQL with a general-purpose programming language. Both ODBC and JDBC expose database capabilities in a standardized way to the application programmer through an application programming interface (API).

An application that interacts with a data source through ODBC or JDBC selects a data source, dynamically loads the corresponding driver, and establishes a connection with the data source.

6.2.1JDBC Architecture

The architecture of JDBC has four main components: the **application**, the **driver manager**, several data source specific **drivers**, and the corresponding data **Sources**.

The **application** initiates and terminates the connection with a data source. It sets transaction boundaries, submits SQL statements, and retrieves the results-----all through a well-defined interface as specified by the JDBC API.

The primary goal of the **driver manager** is to load JDBC drivers and pass JDBC function calls from the application to the correct driver.

The **driver** establishes the connection with the data source.

The **data source** processes commands from the driver and returns the results.

Driversin JDBC are classified into **four types** depending on the architectural relationship between the application and the data source:

- Type I Bridges: This type of driver translates JDBC function calls into function calls of another API that is not native to the DBMS. An example is a JDBC-ODBC bridge; an application can use JDBC calls to access an ODBC compliant data source. The application loads only one driver, the bridge. Bridges have the advantage that it is easy to piggyback the applica.tion onto an existing installation, and no new drivers have to be installed. But using bridges has several drawbacks. The increased number of layers between data source and application affects performance. In addition, the user is limited to the functionality that the ODBC driver supports.
- Type II Direct Translation to the Native API via Non-Java Driver: This type of driver translates JDBC function calls directly into method invocations of the API of one specific data source. The driver is usually written using a combination of C++ and Java; it is dynamically linked and specific to the data source. This architecture performs significantly better than a JDBC-ODBC bridge. One disadvantage is that the database driver that implements the API needs to be installed on each computer that runs the application.
- Type III—Network Bridges: The driver talks over a network to a middleware server that translates the JDBC requests into DBMS-specific method invocations. In this case, the driver on the client site (Le., the network bridge) is not DBMS-specific. The JDBC driver loaded by the application can be quite small, as the only functionality it needs to implement is sending of SQL statements to the middleware server. The middleware server can then use a Type II JDBC driver to connect to the data source.
- Type IV-Direct Translation to the Native API via Java Driver: Instead of calling the DBMS API directly, the driver communicates with the DBMS through Java sockets. In this case, the driver on the client side is written in Java, but it is DBMS-specific. It translates JDBC calls into the native API of the database system. This solution does not require an intermediate layer, and since the implementation is all Java, its performance is usually quite good.

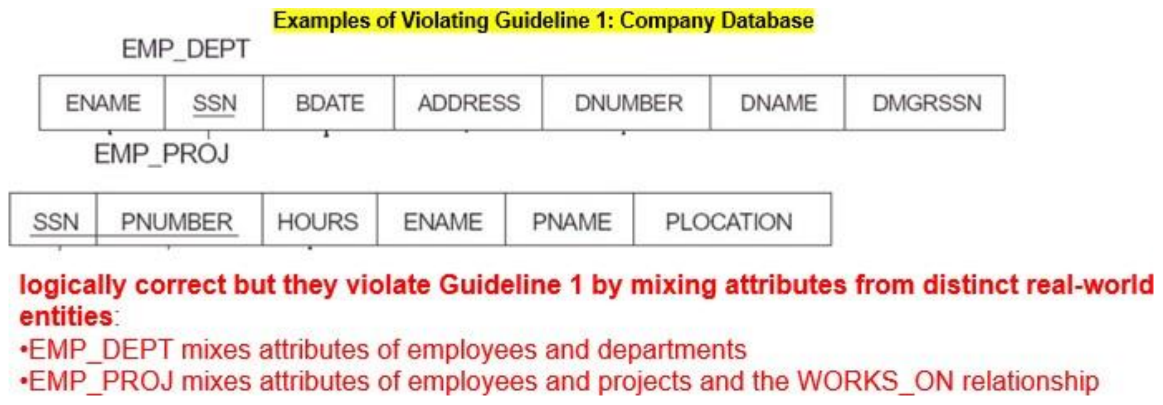
7. a. Discuss informal design guidelines for relational schema.

Four informal guidelines that may be used as measures to determine the quality of relation schema design:
(1) Imparting Clear Semantics to Attributes in Relations

•Design a relation schema so that it is easy to explain its meaning

•Do not combine attributes from multiple entity types and relationship types into a single relation

if a relation schema corresponds to one entity type or one relationship type, it is straightforward to interpret and to explain its meaning. if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.



Examples of Violating Guideline 1: Company Database

EMP_DEPT

ENAME	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

logically correct but they violate Guideline 1 by mixing attributes from distinct real-world entities:

•EMP_DEPT mixes attributes of employees and departments

•EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship

---

(2) Removing Redundant Information in Tuples and Update Anomalies

§One goal of schema design is to minimize the storage space used by the base relations
§Grouping attributes into relation schemas has a significant effect on storage space

update anomalies

- insertion anomalies
- deletion anomalies,
- modification anomalies

(3) Removing NULL Values in Tuples

As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL
§ If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation
§ Using space efficiently and avoiding joins with NULL values are the two overriding criteria that determine whether to include the columns that may have NULLs in a relation or to have a separate relation for those columns with the appropriate key columns

(4) Removing Spurious Tuples
Spurious Tuples are those rows in a table, which occur as a result of joining two tables in wrong manner. They are extra tuples (rows) which might not be required
Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated
§Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

7.b What is normalization? What are its advantages ? discuss 1NF,2NFand 3NF

Normalization of data can be considered a process of analysing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of

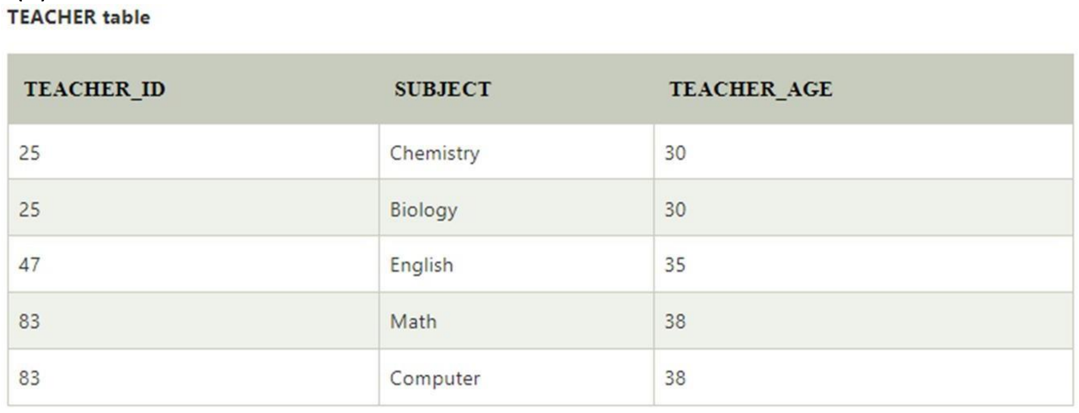
(1) minimizing redundancy and minimizing the insertion, deletion, and update anomalies

Second Normal Form (2NF)

•In the 2NF, relational must be in 1NF.
•In the second normal form, all non-key attributes are fully functional dependent on the primary key
Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

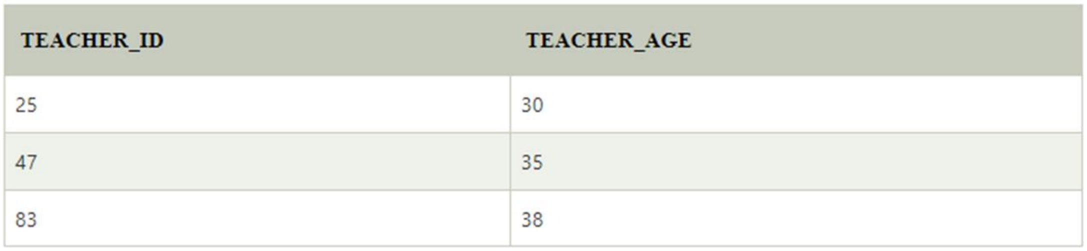
In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:
(2)


TEACHER table

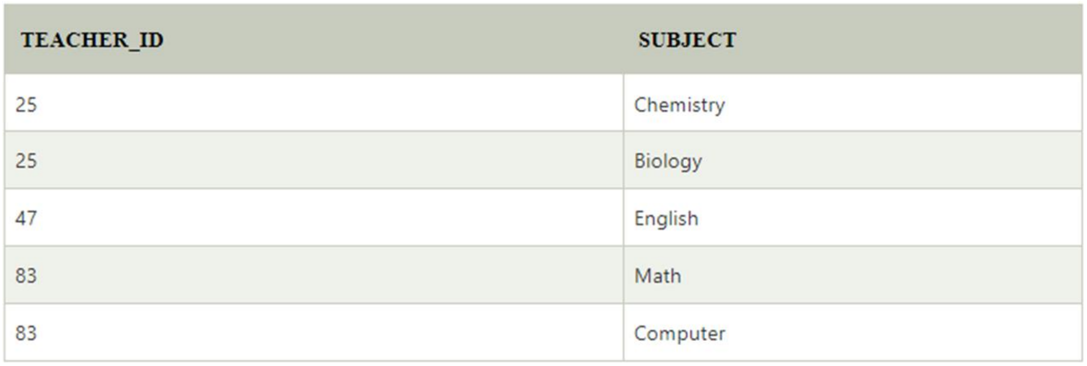
TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

TEACHER_DETAIL table:



TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:



TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

8.b. Discuss the different inference rules for functional dependencies.

Functional dependencies are a concept in database management that describe the relationship between attributes in a relational database. Inference rules for functional dependencies help derive additional dependencies based on a given set of dependencies. Here are some common inference rules for functional dependencies:

Reflexivity Rule:
If Y is a subset of X, then $X \to Y$.
This rule states that if a set of attributes (Y) is functionally dependent on another set of attributes (X), and Y is a subset of X, then $X \to Y$ is also true.

Augmentation Rule:
If $X \to Y$, then $XZ \to YZ$.
This rule allows the extension of a functional dependency by adding attributes to both sides of the dependency.

Transitivity Rule:
If $X \to Y$ and $Y \to Z$, then $X \to Z$.
Transitivity allows the derivation of a new functional dependency based on two existing dependencies.

Union Rule:
If $X \to Y$ and $X \to Z$, then $X \to YZ$.
This rule enables combining two dependencies into a single dependency.

Decomposition Rule:
If $X \to YZ$, then $X \to Y$ and $X \to Z$.
Decomposition allows breaking down a dependency into two or more dependencies.

Pseudotransitivity Rule:
If X → Y and WY → Z, then WX → Z.
Pseudotransitivity helps derive new dependencies when there is a transitive relationship through an intermediary set of attributes.
Trivial Dependency Rule:
If Y is a subset of X, then X → Y is a trivial dependency.
This rule emphasizes that if Y is a subset of X, then X → Y is considered a trivial dependency.
Union-Additivity Rule:
If X → Y and X → Z, then X → YZ.
Similar to the union rule, the union-additivity rule allows combining two dependencies into a single dependency.
Intersection Rule:
If X → Y and X → Z, then X → Y ∩ Z.
This rule allows combining two dependencies into a single dependency by taking the intersection of the dependent attributes.
These rules provide a foundation for reasoning about and inferring functional dependencies in a relational database. They are fundamental for database normalization processes and ensuring data integrity. Understanding these rules helps in designing efficient and well-structured database schemas.

9.b. Discuss the characterizing schedules based on recoverability.
In database management, schedules refer to the order in which transactions are executed. Recoverability is an important property of schedules that ensures that, in the event of a system failure, the database can be restored to a consistent state. There are two types of schedules based on recoverability: recoverable schedules and non-recoverable schedules.
Recoverable Schedules:
A schedule is considered recoverable if, in the event of a transaction failure, it is possible to bring the system back to a consistent state.
The recoverability property ensures that if a transaction T1 reads a value written by another transaction T2, T2 must commit before T1.
In recoverable schedules, the system keeps track of all intermediate states, allowing for a rollback to a consistent state if necessary.
The two common techniques to ensure recoverability are using undo logs (rollback logs) and before-image (write-ahead logging).
Non-Recoverable Schedules:
A schedule is non-recoverable if it does not ensure recoverability in case of a failure.
In non-recoverable schedules, there is no mechanism to undo the effects of committed or uncommitted transactions.
If a transaction T1 reads a value written by another transaction T2, T2 might commit or abort before T1 completes. In the case of an abort by T2, T1 is left with potentially incorrect data.

Non-recoverable schedules are generally not suitable for systems where data consistency is critical.

Example:

Consider two transactions, T1 and T2, with the following operations:

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

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

Recoverable Schedule:

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

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

In this case, T1 reads and writes to A before T2 reads A, and T2 writes to B before T1 reads B. This schedule is recoverable because if T2 aborts, T1 can be rolled back to a consistent state.

Non-Recoverable Schedule:

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

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

If T2 commits before T1 completes, and T1 later fails, there is no mechanism to undo the effects of T2 on A. This schedule is non-recoverable.

Ensuring recoverability is crucial for maintaining the integrity of the database in the face of failures. The use of appropriate logging mechanisms and transaction control protocols helps achieve recoverability in database systems.

10.a. Discuss a lock based concurrency contril issue in DBMS transaction processeing.

It is a mechanism in which a transaction cannot read or write data unless the appropriate lock is acquired. This helps in eliminating the concurrency problem by locking a particular transaction to a particular user. The lock is a variable that denotes those operations that can be executed on the particular data item.

The various types of lock include :

- Binary lock: It ensures that the data item can be in either locked or unlocked state
- Shared Lock: A shared lock is also called read only lock because you don't have permission to update data on the data item. With this lock data item can be easily shared between different transactions. For example, if two teams are working on employee payment accounts, they would be able to access it but wouldn't be able to modify the data on the payment account.
- Exclusive Lock: With exclusive locks, the data items will not be just read but can also be written
- Simplistic Lock Protocol: this lock protocol allows transactions to get lock on every object at the start of operation. Transactions are able to unlock the data item after completing the write operations
- Pre-claiming locking: This protocol evaluates the operations and builds a list of the necessary data items which are required to initiate the execution of the transaction. As soon as the locks are acquired, the execution of transaction takes place. When the operations are over, then all the locks release.

- Starvation: It is the condition where a transaction has to wait for an indefinite period for acquiring a lock.
- Deadlock: It is the condition when two or more processes are waiting for each other to get a resource released

10.b. Describe granularity of data items and Multiple Granularity locking.

The various Concurrency Control schemes have used different methods and every individual Data item is the unit on which synchronization is performed. A certain drawback of this technique is if a transaction Ti needs to access the entire database, and a locking protocol is used, then Ti must lock each item in the database. It is less efficient, it would be simpler if Ti could use a single lock to lock the entire database. But, if it considers the second proposal, this should not in fact overlook certain flaws in the proposed method. Suppose another transaction just needs to access a few data items from a database, so locking the entire database seems to be unnecessary moreover it may cost us a loss of Concurrency, which was our primary goal in the first place. To bargain between Efficiency and Concurrency. Use Granularity.

Let's start by understanding what is meant by Granularity.

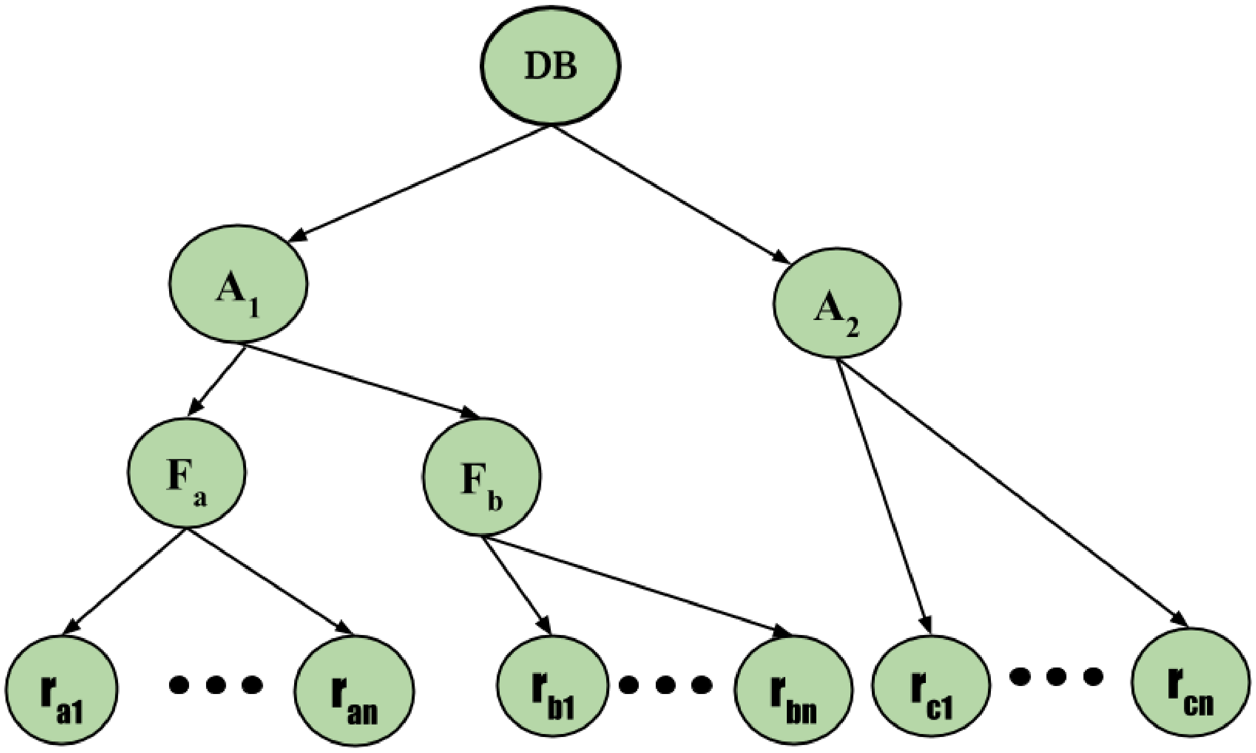
Granularity

It is the size of the data item allowed to lock. Now *Multiple Granularity* means hierarchically breaking up the database into blocks that can be locked and can be tracked what needs to lock and in what fashion. Such a hierarchy can be represented graphically as a tree.

For example, consider the tree, which consists of four levels of nodes. The highest level represents the entire database. Below it is nodes of type area; the database consists of exactly these areas. The area has children nodes which are called files. Every area has those files that are its child nodes. No file can span more than one area.

Finally, each file has child nodes called records. As before, the file consists of exactly those records that are its child nodes, and no record can be present in more than one file. Hence, the levels starting from the top level are:

- database
- area
- file
- record



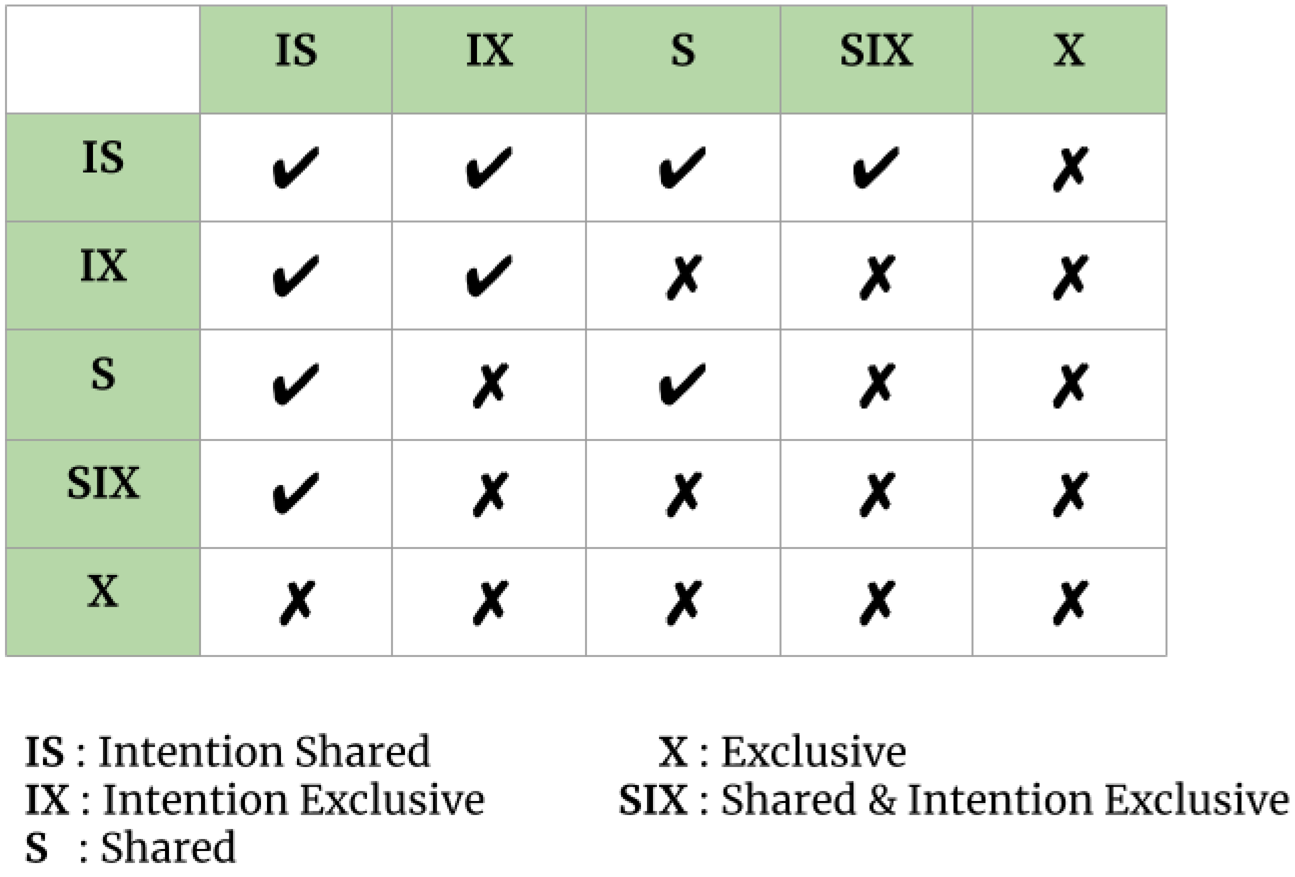
Multi Granularity tree Hiererchy

Consider the above diagram for the example given, each node in the tree can be locked individually. As in the 2-phase locking protocol, it shall use shared and exclusive lock modes. When a transaction locks a node, in either shared or exclusive mode, the transaction also implicitly locks all the descendants of that node in the same lock mode. For example, if transaction Ti gets an explicit lock on file Fc in exclusive mode, then it has an implicit lock in exclusive mode on all the records belonging to that file. It does not need to lock the individual records of Fc explicitly. this is the main difference between Tree-Based Locking and Hierarchical locking for multiple granularities.
Now, with locks on files and records made simple, how does the system determine if the root node can be locked? One possibility is for it to search the entire tree but the solution nullifies the whole purpose of the multiple-granularity locking scheme. A more efficient way to gain this knowledge is to introduce a new lock mode, called *Intention lock mode*.
Intention Mode Lock
In addition to **S** and **X** lock modes, there are three additional lock modes with multiple granularities:

- **Intention-Shared (IS):** explicit locking at a lower level of the tree but only with shared locks.
- **Intention-Exclusive (IX):** explicit locking at a lower level with exclusive or shared locks.

- **Shared & Intention-Exclusive (SIX):** the subtree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive mode locks.

The compatibility matrix for these lock modes are described below:



	IS	IX	S	SIX	X
IS	✔	✔	✔	✔	✗
IX	✔	✔	✗	✗	✗
S	✔	✗	✔	✗	✗
SIX	✔	✗	✗	✗	✗
X	✗	✗	✗	✗	✗

IS : Intention Shared
IX : Intention Exclusive
S : Shared
X : Exclusive
SIX : Shared & Intention Exclusive

Multi Granularity tree Hierarchy

The multiple-granularity locking protocol uses the intention lock modes to ensure serializability. It requires that a transaction Ti that attempts to lock a node must follow these protocols:

- Transaction Ti must follow the lock-compatibility matrix.
- Transaction Ti must lock the root of the tree first, and it can lock it in any mode.
- Transaction Ti can lock a node in S or IS mode only if Ti currently has the parent of the node-locked in either IX or IS mode.
- Transaction Ti can lock a node in X, SIX, or IX mode only if Ti currently has the parent of the node-locked in either IX or SIX modes.
- Transaction Ti can lock a node only if Ti has not previously unlocked any node (i.e., Ti is two-phase).
- Transaction Ti can unlock a node only if Ti currently has none of the children of the node-locked.

Observe that the multiple-granularity protocol requires that locks be acquired in top-down (root-to-leaf) order, whereas locks must be released in bottom-up (leaf to-root) order.
As an illustration of the protocol, consider the tree given above and the transactions:

- Say transaction T1 reads record Ra2 in file Fa. Then, T2 needs to lock the database, area A1, and Fa in IS mode (and in that order), and finally to lock Ra2 in S mode.
- Say transaction T2 modifies record Ra9 in file Fa . Then, T2 needs to lock the database, area A1, and file Fa (and in that order) in IX mode, and at last to lock Ra9 in X mode.
- Say transaction T3 reads all the records in file Fa. Then, T3 needs to lock the database and area A1 (and in that order) in IS mode, and at last to lock Fa in S mode.
- Say transaction T4 reads the entire database. It can do so after locking the database in S mode.

8. a. Explain with an example the Boyce Codd normal form (BCNF)

•BCNF is the advanced version of 3NF. It is stricter than 3NF.

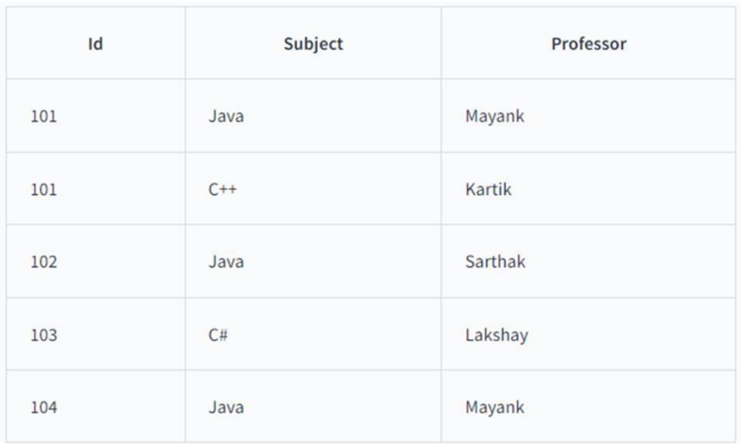
Rules for BCNF in DBMS

A table or relation is said to be in BCNF (Boyce Codd Normal Form) if it satisfies the following two conditions :

•It should satisfy all the conditions of the Third Normal Form (3NF)

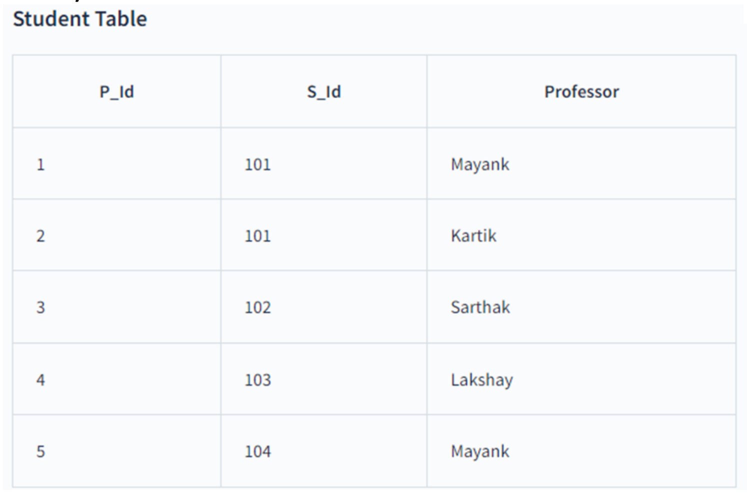
•For any functional dependency (A->B), A should be either the super key or the candidate key. In simple words, it means that A can't be a non-prime attribute if B is given as a prime attribute.

In this example, we have a relation R with three columns: Id, Subject, and Professor.



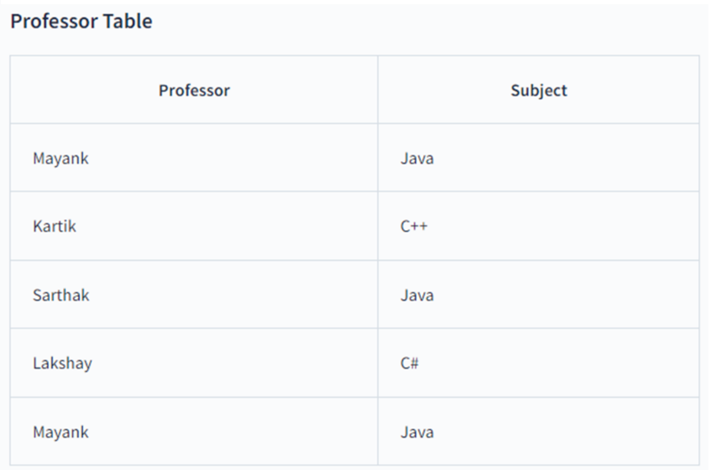
Id	Subject	Professor
101	Java	Mayank
101	C++	Kartik
102	Java	Sarthak
103	C#	Lakshay
104	Java	Mayank

In this Example , we will decompose the table into two tables: the Student table and the Professor table to satisfy the conditions of BCNF.



Student Table

P_Id	S_Id	Professor
1	101	Mayank
2	101	Kartik
3	102	Sarthak
4	103	Lakshay
5	104	Mayank

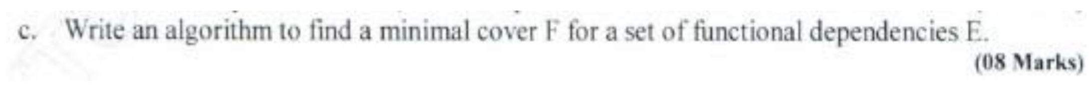


Professor Table

Professor	Subject
Mayank	Java
Kartik	C++
Sarthak	Java
Lakshay	C#
Mayank	Java

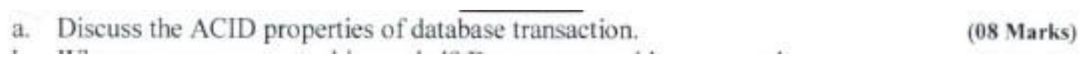
Professor is now the primary key and the prime attribute column, deriving the subject column. Hence, it is in BCNF.

b)



c. Write an algorithm to find a minimal cover F for a set of functional dependencies E. **(08 Marks)**

Q. No. (9)



a. Discuss the ACID properties of database transaction. **(08 Marks)**

Transactions should possess several properties, often called the **<u>ACID properties</u>**

A Atomicity:

a transaction is an atomic unit of processing and it is either performed entirely or not at all.

C Consistency Preservation:

a transaction should be consistency preserving that is it must take the database from one consistent state to another.

I Isolation/Independence:

A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executed concurrently.

D Durability (or Permanency):

if a transaction changes the database and is committed, the changes must never be lost because of any failure.

The **atomicity** property requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity.

The preservation of **consistency** is generally considered to be the responsibility of the programmers who write the database programs or of the DBMS module that enforces integrity constraints.

The **isolation** property is enforced by the concurrency control subsystem of the DBMS. If every transaction does not make its updates (write operations) visible to other transactions until it is committed, one form of isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks

Durability is the responsibility of recovery subsystem.