CMR
INSTITUTE OF
TECHNOLOGY

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

Internal Assessment Test III – September 2023

Sub:	Database Management System							Sub Code:	22MCA21
Date:	26-09-2023	Duration:	90 min's	Max Marks:	50	Sem:	II	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE CO	OBE RBT
1	Define Transaction. Discuss Transaction states with a neat diagram **OR**	[10]	CO3	L3
2	Explain various types of failure that may occur in a system.	[10]	CO3	L3
3	**PART II** Why concurrency control and recovery are needed in DBMS? Explain types of problems that may occur when two simple transaction run concurrently with examples. **OR**	[10]	CO3	L3
4	i. Explain ACID properties of a transaction in detail. ii. When Two operations in a schedule are said to conflict explain with example.	[10]	CO3	L3
5	**PART III** Write a brief note on 2PL with examples. **OR**	[10]	CO5	L3
6	Explain the select and project operation with syntax and examples.	[10]	CO4	L3
7	PART IV Explain Union, intersection and minus operation with examples. **OR**	[10]	CO4	L3
8	Consider the following COMPANY database EMP(Name,SSN,Salary,address, SuperSSN,Gender,Dno) DEPT(DNum,Dname,MgrSSN) PROJECT(Pname,Pnumber,Plocation,Dnum)	[10]	CO4	L3

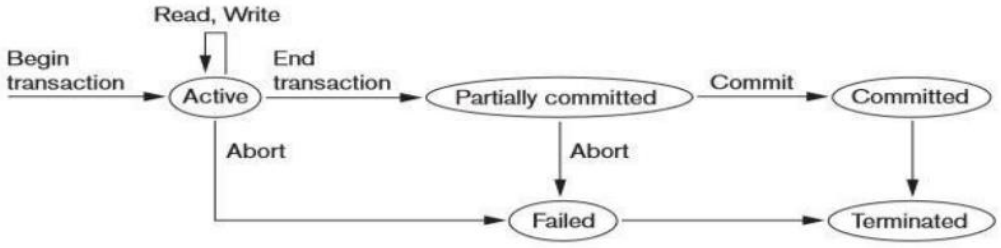
	Write the **relational algebra** queries for the following (i)Retrieve the name, address, salary of employees who work for the Research department. (ii) Find the names of employees who work on all projects controlled by department number 4. iii) Retrieve the SSN of all employees who either in department no :4 or directly supervise an employee who work in dno 4.			
9	PART V Define Domains, Attributes,Tuples, and Relations and also explain the characteristics of relation. **OR**	[10]	CO5	L3
10	Explain the different Relational Model Constraints on databases	[10]	CO5	L3

Solution:

1. Define Transaction. Discuss Transaction states with a neat diagram**.**

A **transaction** is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations—these can include insertion, deletion, modification, or retrieval operations. The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL. One way of specifying the transaction boundaries is by specifying explicit **begin transaction** and **end transaction** statements in an application program; in this case, all database access operations between the two are considered as forming one transaction. A single application program may contain more than one transaction if it contains several transaction boundaries. If the database operations in a transaction do not update the database but only retrieve data, the transaction is called **a read-only transaction**; otherwise it is known as **a read-write transaction**.

State transition diagram illustrating the states for transaction execution.



Transaction States and Additional Operations

A **transaction is an atomic unit of work** that should either be completed in its **entirety or not done** at all. For recovery purposes, the system needs to keep track of when each transaction starts, terminates, and commits or aborts . Therefore, the recovery manager of the DBMS needs to keep track of the following operations:

Figure 21.4 shows a state transition diagram that illustrates how a transaction moves through its execution states .A transaction goes into an **active state** immediately after it starts execution, where it can execute its READ and WRITE operations. When the transaction ends, it moves to the **partially committed** state. At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently (usually by recording changes in the system log). Once this check is successful, the transaction is said to have reached its **commit** point and enters the **committed state**. When a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database, even if a system failure occurs.

However, a transaction can go to the **failed state** if one of the checks fails or if the transaction is **aborte**d during its active state. The transaction may then have to be rolled back to undo the effect of its WRITE operations on the database. The **terminated state** corresponds to the transaction leaving the system. The transaction information that is maintained in system tables while the transaction has been running is removed when the transaction terminates. Failed or aborted transactions may be restarted later—either automatically or after being resubmitted by the user—as brand new transactions.

2. Explain various types of failure that may occur in a system.

Types of Failures. Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

1. **A computer failure (system crash).** A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures—for example, main memory failure.
2. **A transaction or system error.** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error.[3] Additionally, the user may interrupt the transaction during its execution.
3. **Local errors or exception conditions detected by the transaction.** During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. An exception condition,[4] such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception could be programmed in the transaction itself, and in such a case would not be considered as a transaction failure.

4. **Concurrency control enforcement.** The concurrency control method (see Chapter 22) may decide to abort a transaction because it violates serializability (see Section 21.5), or it may abort one or more transactions to resolve a state of deadlock among several transactions (see Section 22.1.3). Transactions aborted because of serializability violations or deadlocks are typically restarted automatically at a later time.
5. **Disk failure.** Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.
6. **Physical problems and catastrophes.** This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

3. Why concurrency control and recovery are needed in DBMS? Explain types of problems that may occur when two simple transaction run concurrently with examples

Why Concurrency Control Is Needed

Several problems can occur when concurrent transactions execute in an uncontrolled manner. We illustrate some of these problems by referring to a much simplified airline reservations database in which a record is stored for each airline flight. Each record includes the number of reserved seats on that flight as a named (uniquely identifiable) data item, among other information. Figure 21.2(a) shows a transaction T1 that 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. Figure 21.2(b) shows a simpler transaction T2 that just reserves M seats on the first flight (X) referenced in transaction T1.2

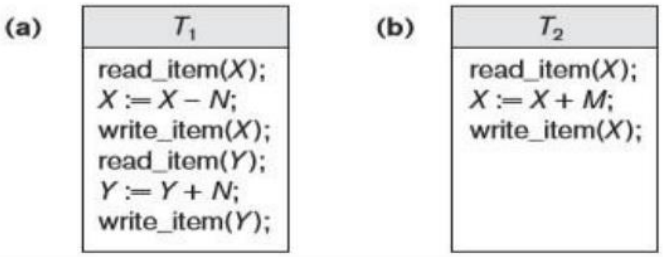


Figure 21.2
Two sample transactions. (a) Transaction T_1. (b) Transaction T_2.

Next we discuss the types of problems we may encounter with these two simple transactions if they run concurrently.

1) **The Lost Update Problem**. This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect. Suppose that transactions T1 and T2 are submitted at approximately the same time, and suppose that their operations are interleaved as shown in Figure 21.3(a); then the final value of item X is incorrect because T2 reads the value of X before T1 changes it in the database, and hence the updated value resulting from T1 is lost. For example, if X = 80 at the start (originally there were 80 reservations on the flight), N = 5 (T1 transfers 5 seat reservations from the flight corresponding to X to the flight corresponding to Y), and M = 4 (T2 reserves 4 seats on X), the final result should be X = 79. However, in the interleaving of operations shown in Figure 21.3(a), it is X = 84 because the update in T1 that removed the five seats from X was lost.

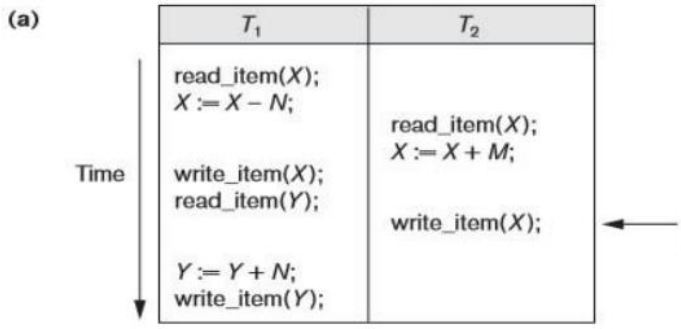
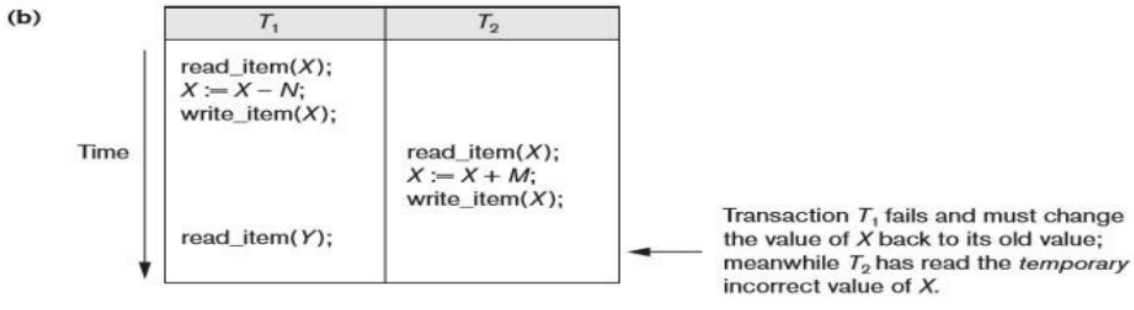


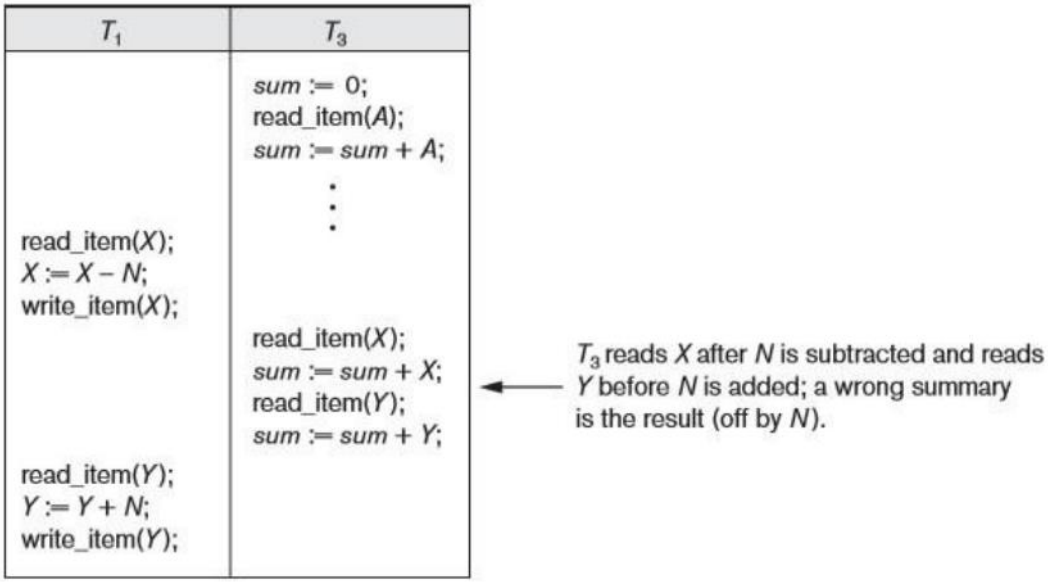
Figure 21.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

2) **The Temporary Update (or Dirty Read) Problem.** This problem occurs when one transaction updates a database item and then the transaction fails for some reason .Meanwhile, the updated item is accessed (read) by another transaction before it is changed back to its original value. Figure 21.3(b) shows an example where T1 updates item X and then fails before completion, so the system must change X back to its original value. Before it can do so, however, transaction T2 reads the temporary value of X, which will not be recorded permanently in the database because of the failure of T1. The value of item X that is read by T2 is called dirty data because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the **dirty read problem**.



3) **The Incorrect Summary Problem.** If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated. For example, suppose that a transaction T3 is calculating the total number of reservations on all the flights; meanwhile, transaction T1 is executing. If the interleaving of operations shown in Figure 21.3(c) occurs, the result of T3 will be off by an amount N because T3 reads the value of X after N seats have been subtracted from it but reads the value of Y before those N seats have been added to it.

(c)



4) **The Unrepeatable Read Problem**. Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction Tbetween the two reads. Hence, T receives different values for its two reads of the same item. This may occur, for example, if during an airline reservation transaction, a customer inquires about seat availability on several flights. When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation ,and it may end up reading a different value for the item.

4. **i.** Explain ACID properties of a transaction in detail.

Desirable Properties of Transactions

Transactions should possess several properties, often called the ACID properties; they should be enforced by the concurrency control and recovery methods of the DBMS.

The following are the **ACID** properties:

- **Atomicity.** A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.
- **Consistency preservation.** A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.
- **Isolation.** A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing

ii. When Two operations in a schedule are said to conflict explain with example.

Two operations in a schedule are said to **conflict** if they satisfy all three of the following conditions:

1. They belong to different transactions;
2. They access the same item *X*; and
3. At least one of the operations is a write_item(*X*).

For example, in schedule S_a, the operations $r_1(X)$ and $w_2(X)$ conflict, as do the operations $r_2(X)$ and $w_1(X)$, and the operations w1(*X*) and w2(*X*). However, the operations r1(*X*) and r2(*X*) do not conflict, since they are both read operations; the operations w2(*X*) and w1(*Y*) do not conflict, because they operate on distinct data items *X* and *Y*; and the operations r1(*X*) and w1(*X*) do not conflict, because they belong to the same transaction.

A schedule *S* of *n* transactions T1, T2, ..., Tn, is said to be a **complete schedule** if the following conditions hold:

1. The operations in *S* are exactly those operations in T1, T2, ..., Tn, including a commit or abort operation as the last operation for each transaction in the schedule.
2. For any pair of operations from the same transaction Ti, their order of appearance in *S* is the same as their order of appearance in Ti.
3. For any two conflicting operations, one of the two must occur before the other in the schedule.

5. Write a brief note on 2PL with examples.

Two-Phase Locking

A transaction is said to follow the two-phase locking protocol if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction. Such a transaction can be divided into **two phases**.

Growing (first) phase: an expanding or growing (first) phase, during which new locks on items can be acquired but none can be released.

shrinking (second) phase : a shrinking (second) phase, during which existing locks can be released but no new locks can be acquired.

If lock conversion is allowed, then **upgrading** of locks (from read-locked to write-locked) must be done during the expanding phase, and **downgrading** of locks (from write-locked to read-locked) must be done in the shrinking phase. Hence, a read_lock(X) operation that downgrades an already held write lock on X can appear only in the shrinking phase.

Variations (Types of two phase locking) of two-phase locking (2PL).

a) Basic, b) Conservative, c) Strict, and d) Rigorous Two-Phase Locking:

There are a number of variations of two-phase locking (2PL). The technique just described is known as **basic 2PL**.

A variation known as **conservative 2PL** (or static 2PL) requires a transaction to lock all the items it accesses before the transaction begins execution, by predeclaring its read-set and write-set. The read-set of a transaction is the set of all items that the transaction reads, and the write-set is the set of all items that it writes. If any of the predeclared items needed cannot be locked, the transaction does not lock any item; instead, it waits until all the items are available for locking. **Conservative 2PL is a deadlock-free protocol.**

In practice, the most popular variation of 2PL is **strict 2PL**, which guarantees strict schedules . In this variation, a transaction T does not release any of its exclusive (write) locks until after it commits or aborts. Hence, no other transaction can read or write an item that is written by T unless T has committed, leading to a strict schedule for recoverability. **Strict 2PL is not deadlock-free .**

A more restrictive variation of strict 2PL is **rigorous 2PL**, which also guarantees strict schedules. In this variation, a transaction T does not release any of its locks (exclusive or shared) until after it commits or aborts, and so it is easier to implement than strict 2PL.

Dealing with Deadlock and Starvation

Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T in the set. Hence, each transaction in the set is in a waiting queue, waiting for one of the other transactions in the set to release the lock on an item. But because the other transaction is also waiting, it will never release the lock.

A simple example is shown in Figure 22.5(a), where the two transactions T1and T2are **deadlocked** in a partial schedule; T1is in the waiting queue for X, which is locked by T2, while T2is in the waiting queue for Y, which is locked by T1. Meanwhile, neither T1nor T2nor any other transaction can access items X and Y.

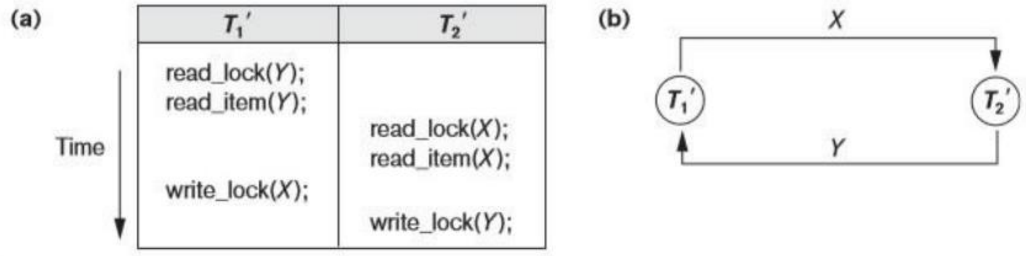


Figure 22.5
Illustrating the deadlock problem. (a) A partial schedule of T_1' and T_2' that is in a state of deadlock. (b) A wait-for graph for the partial schedule in (a).

6. Explain the select and project operation with syntax and examples.

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

σDno=4(EMPLOYEE) σSalary>30000(EMPLOYEE)

- ✓ 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.
- ✓ For example, to select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000:

σ(Dno=4 AND Salary>25000) OR (Dno=5 AND Salary>30000)(EMPLOYEE)

- ✓ 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,

$$\sigma\ (cond1)(\sigma(cond2)(R)) = \sigma(cond2)(\sigma(cond1)(R))$$

The PROJECT Operation

- ✓ The PROJECT operation, selects certain columns from the table and discards the other columns.
- ✓ The result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations: one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns.
- ✓ For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

$$\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$$

- ✓ The general form of the PROJECT operation is

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

where (pi) is the symbol used to represent the PROJECT operation, and is the desired sublist of attributes from the attributes of relation R.

- ✓ The result of the PROJECT operation has only the attributes specified in in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>.
- ✓ The PROJECT operation removes any duplicate tuples, so the result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as duplicate elimination.

7. Explain Union, intersection and minus operation with examples.

The UNION, INTERSECTION, and MINUS Operations

■ UNION: The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

■ INTERSECTION: The result of this operation, denoted by R ∩ S, is a relation that includes all tuples that are in both R and S.

■ SET DIFFERENCE (or MINUS): The result of this operation, denoted by R – S, is a relation that includes all tuples that are in R but not in S.

- ✓ These are binary operations; that is, each is applied to two sets (of tuples).
- ✓ When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples; this condition has been called union compatibility or type compatibility.
- ✓ Two relations R(A1, A2, … , An) and S(B1, B2, … , Bn) are said to be union compatible (or type compatible) if they have the same degree n and if dom(Ai) = dom(Bi) for $1 \leq i \leq n$. This means that the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.
- ✓ For example, to retrieve the Social Security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5,

$$\begin{aligned} &DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE) \\ &RESULT1 \leftarrow \pi_{Ssn}(DEP5\_EMPS) \\ &RESULT2(Ssn) \leftarrow \pi_{Super\_ssn}(DEP5\_EMPS) \\ &RESULT \leftarrow RESULT1 \cup RESULT2 \end{aligned}$$

- ✓ Both UNION and INTERSECTION are commutative operations; that is,

$$R \cup S = S \cup R \text{ and } R \cap S = S \cap R$$

- ✓ Both UNION and INTERSECTION can be treated as n-ary operations applicable to any number of relations because both are also associative operations; that is,

$$R \cup (S \cup T) = (R \cup S) \cup T \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

- ✓ The MINUS operation is not commutative; that is, in general, $R - S \neq S - R$
- ✓ The INTERSECTION can be expressed in terms of union and set difference as follows:

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

8. Consider the following COMPANY database

EMP(Name,SSN,Salary,address, SuperSSN,Gender,Dno)

DEPT(DNum,Dname,MgrSSN)

PROJECT(Pname,Pnumber,Plocation,Dnum)

Write the **relational algebra** queries for the following

(i)Retrieve the name, address, salary of employees who work for the Research department.

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

$RESEARCH\_DEPT \leftarrow \sigma_{Dname='Research'}(DEPARTMENT)$

$RESEARCH\_EMPS \leftarrow (RESEARCH\_DEPT \bowtie_{Dnumber=Dno} EMPLOYEE)$

$RESULT \leftarrow \pi_{Fname, Lname, Address}(RESEARCH\_EMPS)$

As a single expression, this query becomes

$\pi_{Fname, Lname, Address}(\sigma_{Dname='Research'}(DEPARTMENT \bowtie_{Dnumber=Dno} EMPLOYEE))$

(ii) Find the names of employees who work on all projects controlled by department number 4.

Query 3. Find the names of employees who work on *all* the projects controlled by department number 5.

$DEPT5\_PROJS(Pno) \leftarrow \pi_{Pnumber}(\sigma_{Dnum=5}(PROJECT))$

$EMP\_PROJ(Ssn, Pno) \leftarrow \pi_{Essn, Pno}(WORKS\_ON)$

$RESULT\_EMP\_SSNS \leftarrow EMP\_PROJ + DEPT5\_PROJS$

$RESULT \leftarrow \pi_{Lname, Fname}(RESULT\_EMP\_SSNS * EMPLOYEE)$

iii) Retrieve the SSN of all employees who either in department no :4 or directly supervise an employee who work in dno 4.

For example, to retrieve the Social Security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5,

$DEP5\_EMPS \leftarrow \sigma Dno=5(EMPLOYEE)$
$RESULT1 \leftarrow \pi Ssn(DEP5\_EMPS)$
$RESULT2(Ssn) \leftarrow \pi Super\_ssn(DEP5\_EMPS)$
$RESULT \leftarrow RESULT1 \cup RESULT2$

9. Define Domains, Attributes,Tuples, and Relations and also explain the characteristics of relation.

- ✓ The relational model represents the database as a collection of ***relations***.

Informally, each relation resembles a table of values or, to some extent, a flat file of records

- ✓ A relation is thought of as a **table** of values, each row in the table represents a collection of related data values.
- ✓ A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.
- ✓ In the formal relational model terminology,
  a row →a tuple, a column header →an attribute, and the table →a relation. The data type describing the types of values that can appear in each column is represented by a domain of possible values.

Domains, Attributes, Tuples, and Relations

- ✓ A domain D is a set of **atomic values.** By atomic means each value in the domain is **indivisible** in formal relational model. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
- ✓ Some examples of domains follow:

USA_phone_number: string of digits of length ten
SSN: string of digits of length nine0

Name: string of characters beginning with an upper case letter
GPA: a real number between 0.0 and 4.0
Sex: a member of the set { female, male }
Dept_Code: a member of the set { CMPS, MATH, ENGL, PHYS, PSYC, ... }

- ✓ A **relation schema R**, denoted by R(A1, A2, … , An), is made up of a relation name R and a list of attributes, A1, A2, … , An.
- ✓ **Attribute:** Ai is the name of a role played by some domain D in the relation schema R. D is called the domain of Ai and is denoted by dom(Ai).
- ✓ **Tuple**: A tuple is a mapping from attributes to values drawn from the respective domains of those attributes. A tuple is intended to describe some entity (or relationship between entities) in the miniworld.
- ✓ R is called the name of this relation.
- ✓ The **degree (or arity) of a relation** is the number of attributes n of its relation schema.
- ✓ A relation of degree seven, which stores information about university students, would contain seven attributes describing each student as follows:
  STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)
- ✓ **Relational Database**: A collection of **relations**, each one consistent with its specified relational schema.
- ✓ A **relation (or relation state)** r of the relation schema R(A1, A2, … , An), also denoted by r(R), is a set of n-tuples r = {t1, t2, … , tm}. Each n-tuple t is an ordered list of n values t =<v_1,v_2…,v_n>

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

10. Explain the different Relational Model Constraints on databases.

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:
  $t1[SK] \neq t2[SK]$

- ✓ A superkey SK specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value for SK.
- ✓ A key k of a relation schema R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K′ that is not a superkey of R any more.

  Hence, a key satisfies two properties:

  1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This uniqueness property also applies to a superkey.

  2. It is a minimal superkey—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint hold. This minimality property is required for a key but is optional for a superkey.

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.

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**.