Scheme of Evaluation

Internal Assessment Test III – MAR 2024

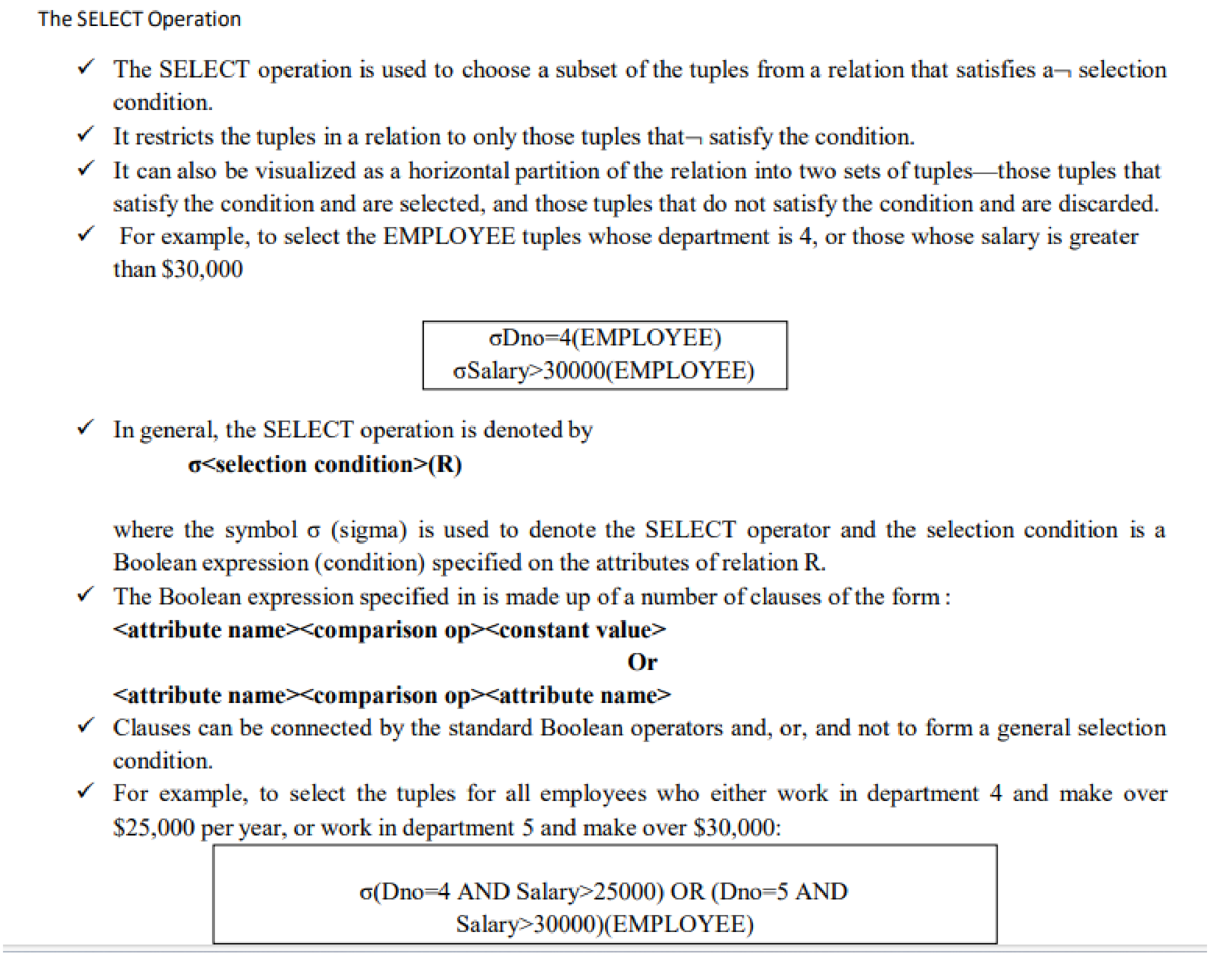
Sub:	**Database Management System**							**Sub Code:**	**21CS53**
Date:	14-03-2024	Duration:	90 mins	Max Marks:	50	**Sem:**	V	**Branch:**	**ISE**

	Answer any FIVE FULL QUESTIONS	MARKS	Marks Distribution
1	Explain the select and project operation with syntax and examples. .What is union compatibility? How union differ from cross product.	[10]	3 3 4
2	Define Domains, Attributes, Tuples, and Relations and also explain the characteristics of relation.	[10]	5 5
3	Define Transaction. Discuss Transaction states with a neat diagram and ACID properties of a transaction in detail	[10]	3 7
4	Explain about Binary Lock and Shared/Exclusive Lock	[10]	5, 5
5	Why concurrency control and recovery are needed in DBMS? Explain types of problems that may occur when two simple transaction run concurrently with examples.	[10]	5, 5
6	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. (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.	[10]	3 3 4

SOLUTIONS

Question 1- Explain the select and project operation with syntax and examples. What is union compatibility? How union differ from cross product.

Solution 1



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,

$$\boldsymbol{\sigma\ (\mathrm{cond1})(\sigma(\mathrm{cond2})(R)) = \sigma(\mathrm{cond2})(\sigma(\mathrm{cond1})(R))}$$

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:

$$\boxed{\pi\text{Lname, Fname, Salary(EMPLOYEE)}}$$

- ✓ The general form of the PROJECT operation is

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

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.

Question 2- Define Domains, Attributes, Tuples, and Relations and also explain the characteristics of relation.

SOLUTION 2

- ✓ 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.**

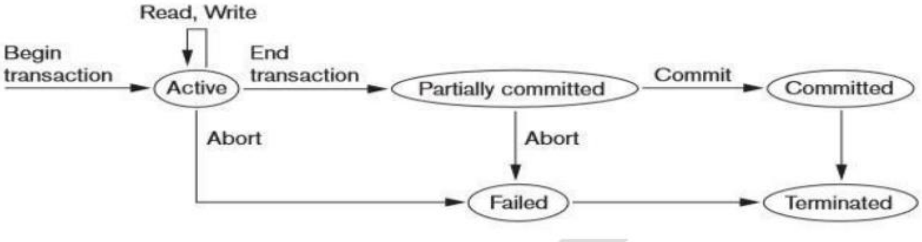
4. **Interpretation (Meaning) of a Relation**
   - ✓ Each tuple in the relation can then be interpreted as a **fact** or a particular instance of the assertion.
   - ✓ Each relation can be viewed as a **predicate** and each tuple in that relation can be viewed as an assertion for which that predicate 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

Question 3- Define Transaction. Discuss Transaction states with a neat diagram and ACID properties of a transaction in detail

SOLUTION 3

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.

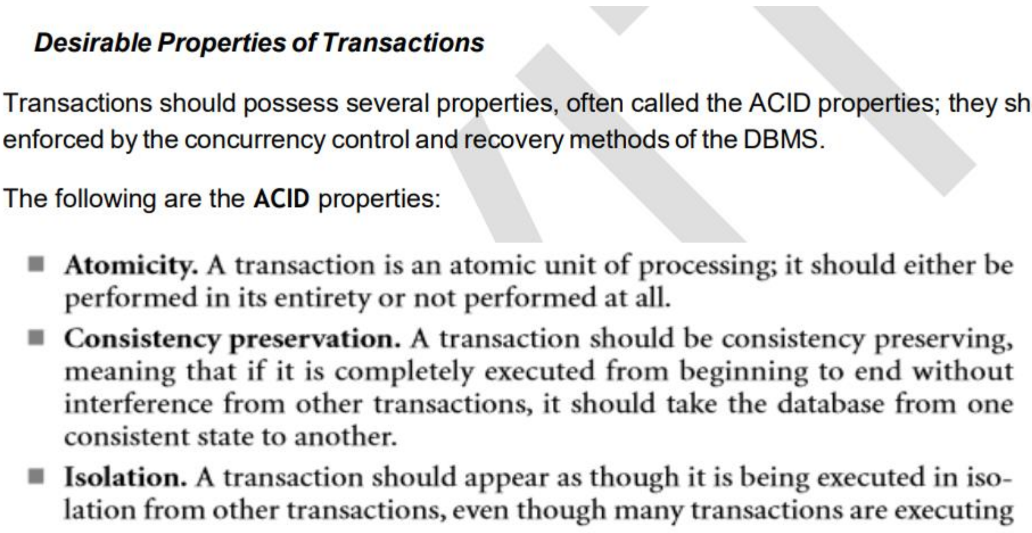


Transaction States and Additional Operations

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 **aborted** 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.



Desirable Properties of Transactions

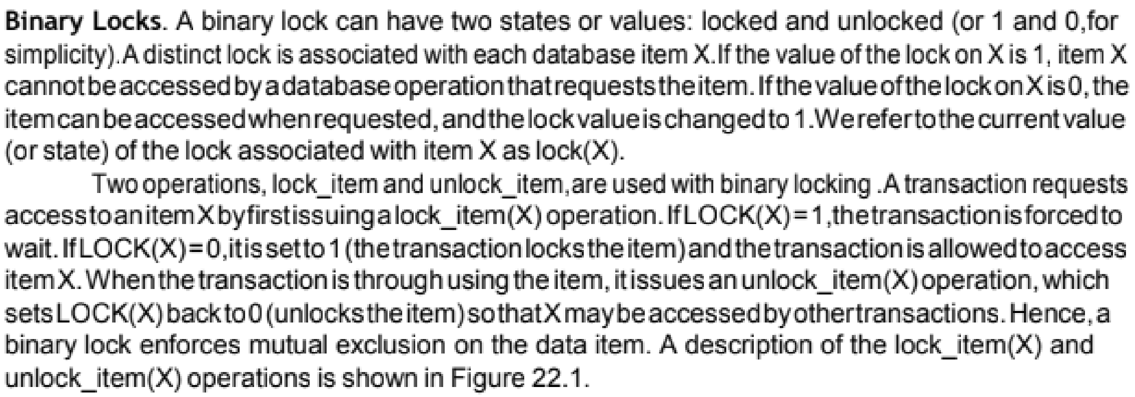
Transactions should possess several properties, often called the ACID properties; they should be enforced by the concurrency control and recovery methods of the DBMS.

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

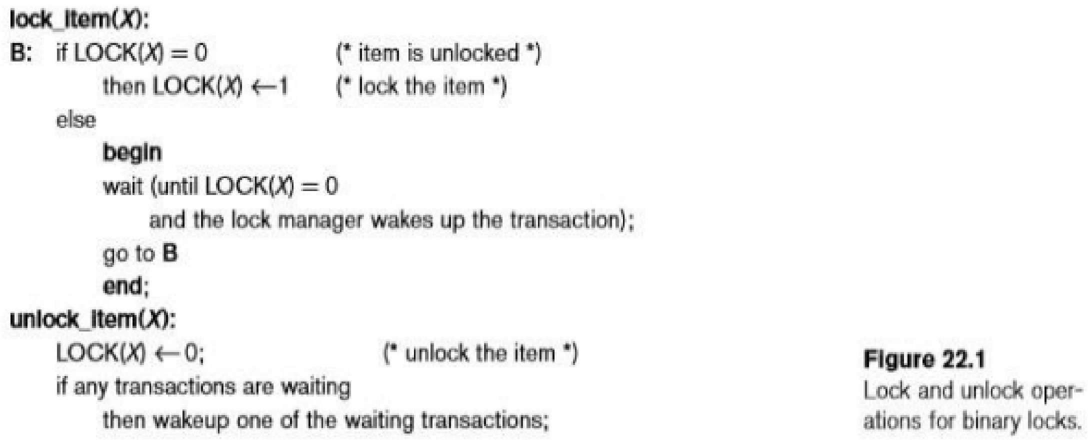
Question 4- Explain about Binary Lock and Shared/Exclusive Lock

SOLUTION 4



Binary Locks. A binary lock can have two states or values: locked and unlocked (or 1 and 0,for simplicity).A distinct lock is associated with each database item X.If the value of the lock on X is 1, item X cannot be accessed by a database operation that requests the item. If the value of the lock on X is 0, the item can be accessed when requested, and the lock value is changed to 1.We refer to the current value (or state) of the lock associated with item X as lock(X).

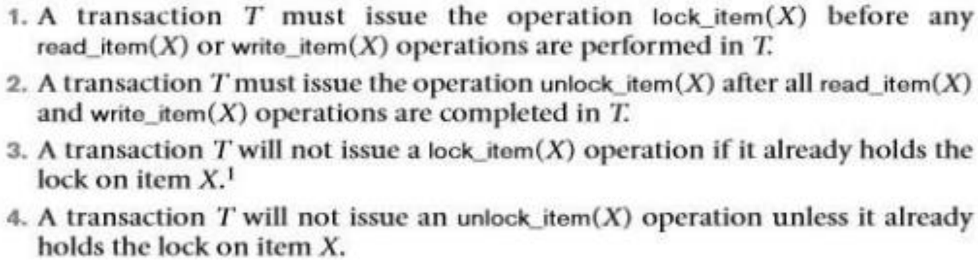
Two operations, lock_item and unlock_item,are used with binary locking .A transaction requests access to an item X by first issuing a lock_item(X) operation. If LOCK(X)=1,the transaction is forced to wait. If LOCK(X)=0,it is set to 1 (the transaction locks the item) and the transaction is allowed to access item X. When the transaction is through using the item, it issues an unlock_item(X) operation, which sets LOCK(X) back to 0 (unlocks the item) so that X may be accessed by other transactions. Hence, a binary lock enforces mutual exclusion on the data item. A description of the lock_item(X) and unlock_item(X) operations is shown in Figure 22.1.



```
lock_item(X):
B:  if LOCK(X) = 0              (* item is unlocked *)
        then LOCK(X) ←1         (* lock the item *)
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ←0;                 (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

Figure 22.1
Lock and unlock operations for binary locks.

If the simple binary locking scheme described here is used, every transaction must obey the following rules:



1. A transaction T must issue the operation lock_item(X) before any read_item(X) or write_item(X) operations are performed in T.
2. A transaction T must issue the operation unlock_item(X) after all read_item(X) and write_item(X) operations are completed in T.
3. A transaction T will not issue a lock_item(X) operation if it already holds the lock on item X.[1]
4. A transaction T will not issue an unlock_item(X) operation unless it already holds the lock on item X.

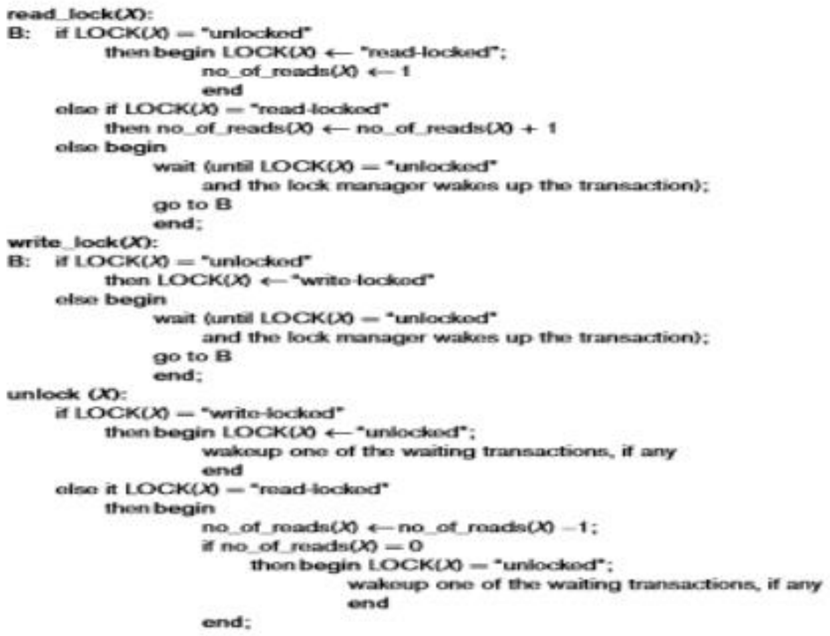
Shared/Exclusive (or Read/Write) Locks: The preceding binary locking scheme is too restrictive for database items because at most, one transaction can hold a lock on a given item. We should allow several transactions to access the same item X if they all access X for reading purposes only. This is because read operations on the same item by different transactions are not conflicting. However,if a transaction is to write an item X,it must have exclusive access to X.

For this purpose, a different type of lock called a **multiple-mode** lock is used. In this scheme—called **shared/exclusive** or **read/write** locks—there are **three locking operations: read_lock(X), write_lock(X), and unlock(X).** A lock associated with an item X, LOCK(X), now has three possible states: read-locked, write-locked, or unlocked. A read-locked item is also called share-locked because other transactions are allowed to read the item, whereas a write-locked item is called exclusive-locked because a single transaction exclusively holds the lock on the item.

When we use the shared/exclusive locking scheme, the system must enforce the following rules:

1. A transaction T must issue the operation read_lock(X) or write_lock(X) before any read_item(X) operation is performed in T.
2. A transaction T must issue the operation write_lock(X) before any write_item(X) operation is performed in T.
3. A transaction T must issue the operation unlock(X) after all read_item(X) and write_item(X) operations are completed in T.
4. A transaction T will not issue a read_lock(X) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X. This rule may be relaxed,as we discuss shortly.
5. A transaction T will not issue a write_lock(X) operation if it already holds a read (shared) lock or write (exclusive) lock on item X.This rule may also be relaxed,as we discuss shortly.
6. A transaction T will not issue an unlock(X) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item X.

Conversion of Locks. Sometimes it is desirable to relax conditions 4 and 5 in the preceding list in order to allow **lock conversion**; that is, a transaction that already holds a lock on item X is allowed under certain conditions to **convert the lock from one locked state to another**. For example, it is possible for a transaction T to issue a read_lock(X) and then later to **upgrade** the lock by issuing a write_lock(X) operation. If T is the only transaction holding a read lock on X at the time it issues the write_lock(X) operation, the lock can be upgraded; otherwise, the transaction must wait. It is also possible for a transaction T to issue a write_lock(X) and then later to **downgrade** the lock by issuing a read_lock(X) operation



```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
                no_of_reads(X) ← 1
                end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else it LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) −1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```

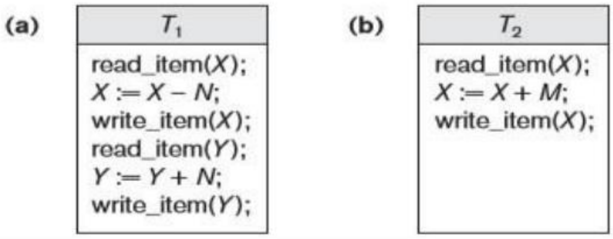
Figure 22.2
Locking and unlocking operations for two-mode (read-write or shared-exclusive) locks.

Question 5- Why concurrency control and recovery are needed in DBMS? Explain types of problems that may occur when two simple transaction run concurrently with examples.

SOLUTION 5

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



(a)

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

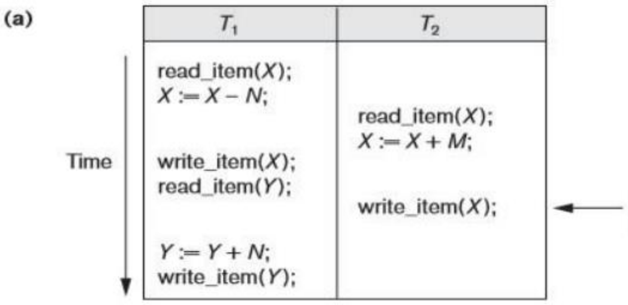
T_2

read_item(X);
$X := X + M$;
write_item(X);

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.



T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;

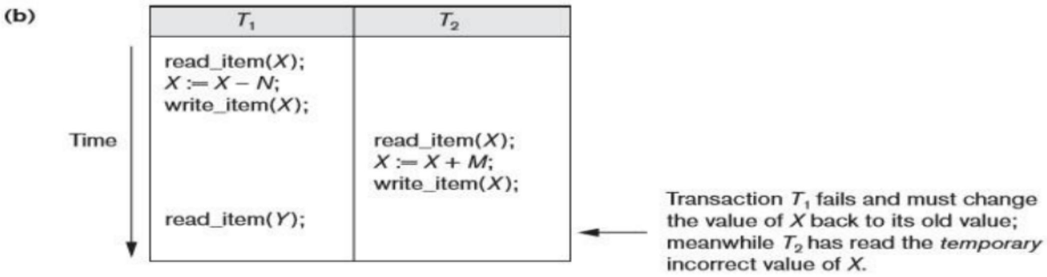
write_item(X);
read_item(Y);	

	write_item(X);
$Y := Y + N$; write_item(Y);	

Item X has an incorrect value because its update by T_1 is *lost* (overwritten).

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**.



T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

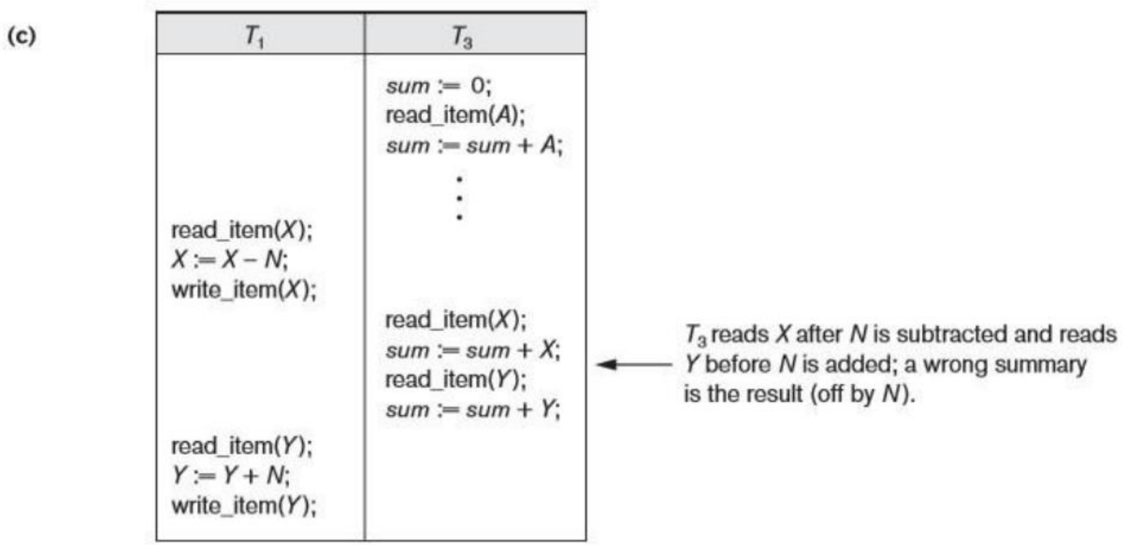
	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);	

Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the *temporary* incorrect value of X.

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)



T_1	T_3

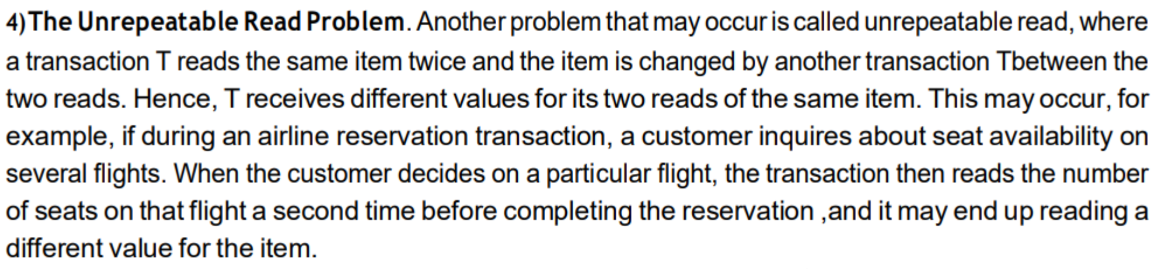
	sum := 0;
	read_item(*A*);
	sum := *sum* + *A*;
	⋮

read_item(*X*);
X := *X* − *N*;
write_item(*X*);	

	read_item(*X*);
	sum := *sum* + *X*;
	read_item(*Y*);
	sum := *sum* + *Y*;

read_item(*Y*);
Y := *Y* + *N*;
write_item(*Y*);	

T_3 reads *X* after *N* is subtracted and reads *Y* before *N* is added; a wrong summary is the result (off by *N*).



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.

Question 6- Consider the following COMPANY database

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

DEPT(DNum,Dname,MgrSSN)

PROJECT(Pname,Pnumber,Plocation,Dnum)

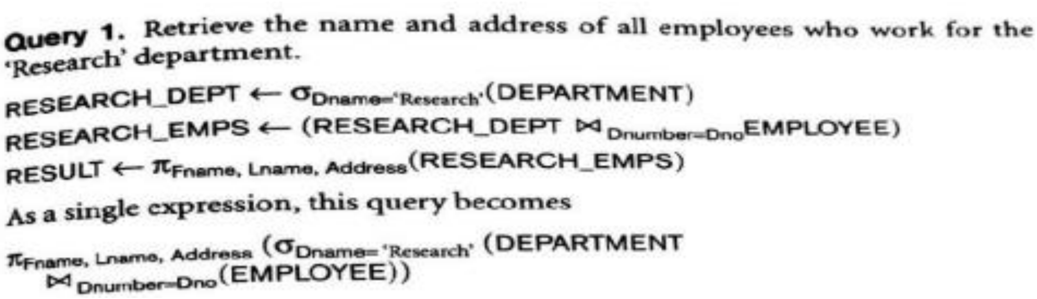
Write the relational algebra queries for the following

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

(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.

SOLUTION 6



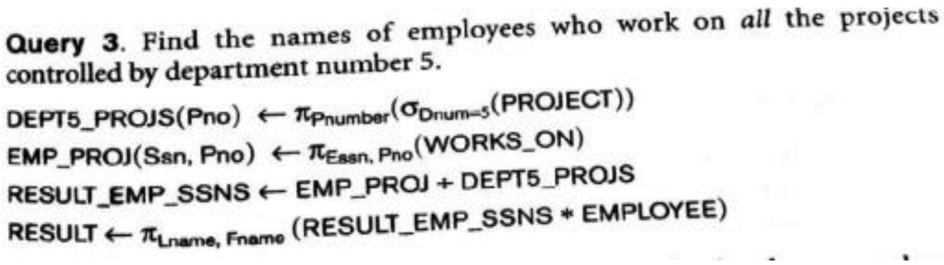
Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

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

As a single expression, this query becomes

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

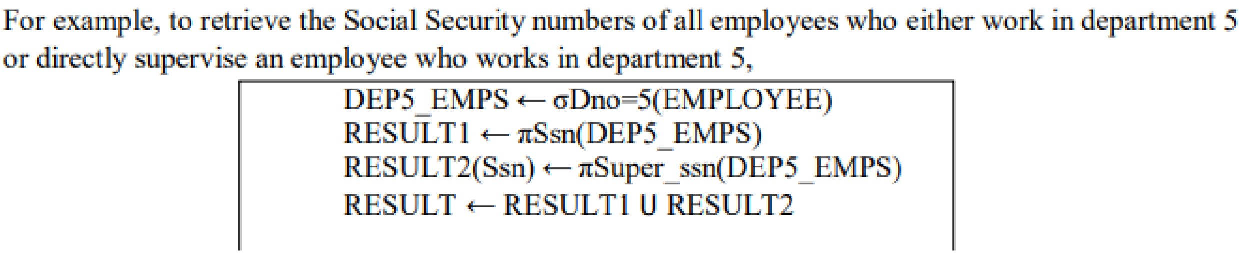
(ii) Find the names of employees who work on all projects controlled by department number 4.



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

DEPT5_PROJS(Pno) ← $\pi_{Pnumber}(\sigma_{Dnum=5}(PROJECT))$
EMP_PROJ(Ssn, Pno) ← $\pi_{Essn, Pno}$(WORKS_ON)
RESULT_EMP_SSNS ← EMP_PROJ ÷ DEPT5_PROJS
RESULT ← $\pi_{Lname, Fname}$(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 ← σDno=5(EMPLOYEE)
RESULT1 ← πSsn(DEP5_EMPS)
RESULT2(Ssn) ← πSuper_ssn(DEP5_EMPS)
RESULT ← RESULT1 U RESULT2