DBMS 2015-16 Solution

Q1. Attempt all Parts

a)

i. Data Inconsistency
ii. Data Redundancy
iii. Limited data sharing
iv. Integrity Problem
v. Security Problem

b. **Simple attribute** − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

Composite attribute − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name

c) Referential integrity refers to the accuracy and consistency of data within a relationship. In relationships, data is linked between two or more tables. This is achieved by having the foreign key (in the associated table) reference a primary key value (in the primary – or parent – table)

d) Data Definition Language (DDL) statements are used to define the database structure or schema. the lists of tasks that come under DDL: create, alter, drop, rename, truncate etc.

Data Manipulation Language: A language that offers a set of operations to support the fundamental data manipulation operations on the data held in the database. Data Manipulation Language (DML) statements are used to manage data within schema objects. DML tasks are: insert, select, update, delete.

e) Functional Dependency (FD) determines the relation of one attribute to another attribute in a database management system (DBMS) system. The functional dependency of X on Y is represented by X → Y for ex: emp_no->emp_name(employee name is dependent on employee id)

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. This dependence can be represented like this: car_model -> maf_year and car_model-> colour

f) Multiversion Concurrency Control (MVCC) is a method of controlling consistency of data accessed by **multiple** users concurrently. MVCC implements the snapshot isolation guarantee which ensures that each transaction always sees a consistent snapshot of data.

g) Pitfalls of lock-based protocol are: The potential for deadlock exists in most **locking protocols**. Deadlocks are a necessary evil. Starvation is also possible if concurrency control manager is badly designed.

h) A Multimedia database (MMDB) is a collection of related for multimedia data. The multimedia data include one or more primary media data types such as text, images, graphic objects (including drawings, sketches and illustrations) animation sequences, audio and video.

i) Two table are said to be **union compatible** if both the table have same number of attributes (column) and corresponding attributes have the same data type (int,char,float,date etc.). Corresponding attributes means first attributes of both relations, then second and so on.

j) There are three types of Data **Anomalies**: Update **Anomalies**, Insertion **Anomalies**, and Deletion **Anomalies**.

Q2 Relational Algebra Queries

i. πcode(registered)

ii. πtitle (Student $\bowtie$Registered $\bowtie$Course)

iii. πcode ((πcode (Course) -πcode (Registered)) $\bowtie$Course)

iv. πtitle ((πcode (Course) -πcode (Registered)) $\bowtie$Course)

v. πname,title (Student $\bowtie$Registered $\bowtie$Course)

vi. πssn (Student $\bowtie$Registered $\bowtie$(σ title='Database Systems' Course)) $\cap\pi$ssn (Student $\bowtie$Registered $\bowtie$(σtitle='Analysis of Algorithms' Course))

vii. πssn (Student $\bowtie$Registered $\bowtie$(σ title='Database Systems' Course)) $\cap\pi$ssn (Student $\bowtie$Registered $\bowtie$(σtitle='Analysis of Algorithms' Course))

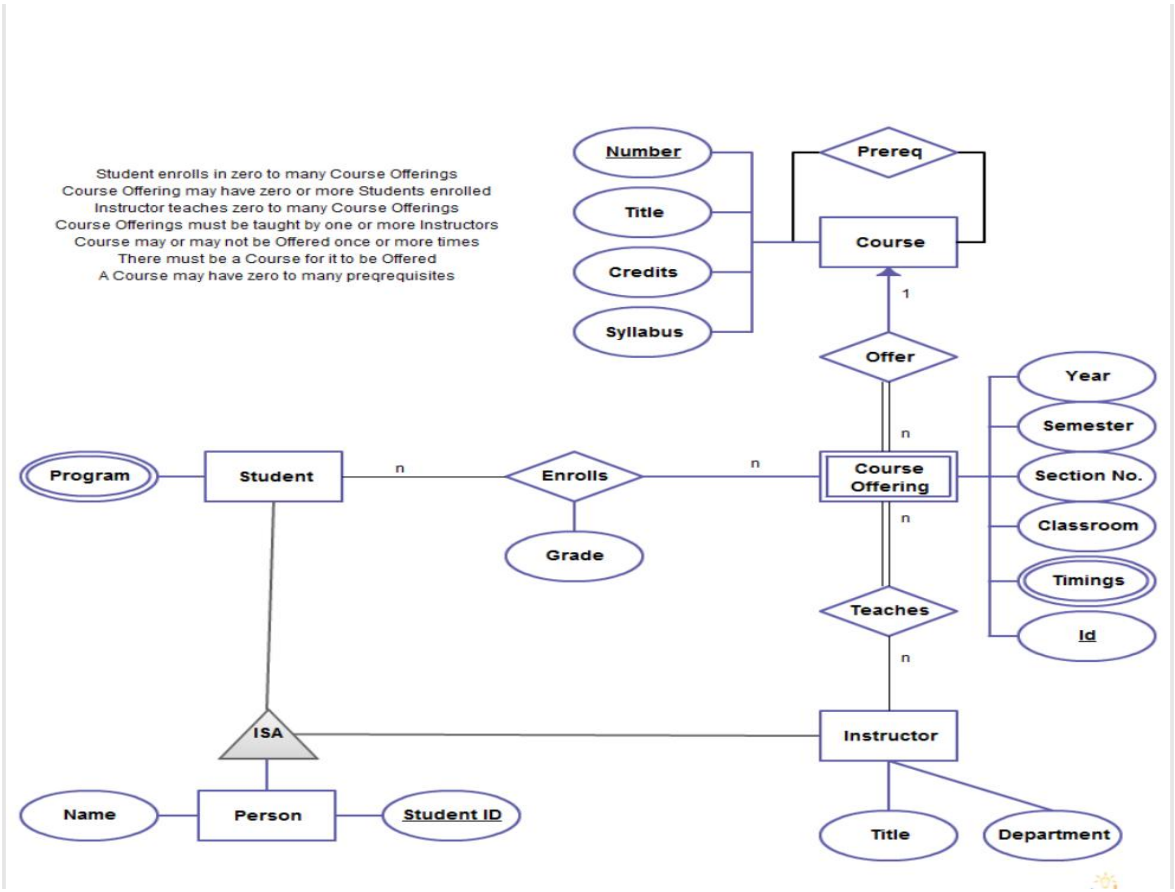
viii. πname (Student ⋈Registered ⋈(σ title=’Database Systems’ Course)) ∩πssn (Student ⋈Registered ⋈(σtitle=’Analysis of Algorithms’ Course))

ix. πcode, ssn (Registered) / πssn (Student)

x. ssn_ecmp<-πssn(σmajor=’ECMP’(Student))

result<-πcode((ssn_ecmp⋈Registered)σssn_ecmp.ssn=Registered.ssn)

Q3. E-R diagram for registrar’s office


Student enrolls in zero to many Course Offerings
Course Offering may have zero or more Students enrolled
Instructor teaches zero to many Course Offerings
Course Offerings must be taught by one or more Instructors
Course may or may not be Offered once or more times
There must be a Course for it to be Offered
A Course may have zero to many preqrequisites
Number
Title
Credits
Syllabus
Prereq
Course
1
Offer
n
Year
Semester
Section No.
Classroom
Timings
Id
Program
Student
n
Enrolls
n
Course Offering
Grade
n
Teaches
n
ISA
Instructor
Name
Person
Student ID
Title
Department

Q4. What do you mean by a key to a relation? Explain the differences between super key, candidate key and primary key.

KEY is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table. Key is also helpful for finding unique record or row from the table. Database key is also helpful for finding unique record or row from the table.

Example:

Employee ID	FirstName	LastName
11	Andrew	Johnson
22	Tom	Wood
33	Alex	Hale

In the above-given example, employee ID is a primary key because it uniquely identifies an employee record. In this table, no other employee can have the same employee ID.

S.No	Super Key	Primary Key	Candidate Key
1.	Super Key is an attribute (or set of attributes) that is used to uniquely identifies all attributes in a relation.	Primary Key is a minimal set of attributes (or set of attributes) that is used to uniquely identifies all attributes in a relation.	Candidate Key is a proper subset of a super key.
2.	All super keys can't be primary keys or candidate keys.	Primary key is a minimal super key.	But all candidate keys are super keys.

3.	Super key’s attributes can contain NULL values.	Primary key’s attributes cannot contain NULL values.	Candidate key’s attributes can also contain NULL values.
4.	In a relation, number of super keys are more than number of candidate keys or primary keys.	While in a relation, number of primary keys are less than number of super keys.	While in a relation, number of candidate keys are less than number of super keys.

A superkey is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

Example:

EmpSSN	EmpNum	Empname
9812345098	AB05	Shown
9876512345	AB06	Roslyn
199937890	AB07	James

In the above-given example, EmpSSN and EmpNum name are superkeys.

PRIMARY KEY is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key. **Example:** StudID is a Primary Key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

CANDIDATE KEY is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key. Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

Q5. Define functional dependency. What do you mean by lossless decomposition? Explain with suitable example how functional dependencies can be used to show that decompositions are lossless.

Functional Dependency (FD) determines the relation of one attribute to another attribute in a database management system (DBMS) system. The functional dependency of X on Y is represented by X → Y for ex: emp_no->emp_name(employee name is dependent on employee id)

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations.

Lossless Join Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation

If we decompose a relation R into relations R1 and R2,

- Decomposition is lossy if $R1 \bowtie R2 \supset R$
- Decomposition is lossless if $R1 \bowtie R2 = R$

To check for lossless join decomposition using FD set, following conditions must hold:

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.
   $Att(R1) \cup Att(R2) = Att(R)$
2. Intersection of Attributes of R1 and R2 must not be NULL.
   $Att(R1) \cap Att(R2) \neq \Phi$
3. Common attribute must be a key for at least one relation (R1 or R2)
   $Att(R1) \cap Att(R2)$ -> $Att(R1)$ or $Att(R1) \cap Att(R2)$ -> $Att(R2)$

For Example, A relation R (A, B, C, D) with FD set{A->BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:

1. First condition holds true as Att(R1) U Att(R2) = (ABC) U (AD) = (ABCD) = Att(R).
2. Second condition holds true as Att(R1) ∩ Att(R2) = (ABC) ∩ (AD) ≠ Φ
3. Third condition holds true as Att(R1) ∩ Att(R2) = A is a key of R1(ABC) because A->BC is given.
   As all the three conditions hold, So decomposition of a relation is a lossless join decomposition.

Q6. Define normal forms. List the definitions of first, second and third normal forms. Explain BCNF with a suitable example.

Normalization is the process of organizing the data in the database. It is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization divides the larger table into the smaller table and links them using relationship. The normal form is used to reduce redundancy from the database table.

- **First Normal Form**-A relation is in 1NF if it contains an atomic value. Each record needs to be unique.
- **Second Normal Form**- A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
- **Third Normal Form**- A relation will be in 3NF if it is in 2NF and no transition dependency exists.
- **Boyce Codd Normal**- Form BCNF is the advance version of 3NF. It is stricter than 3NF.A table is in BCNF if every functional dependency X → Y, X is the super key of the table. In other words, For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

  Example: Let's assume there is a company where employees work in more than one department.
  EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300

364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table, Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Section-B

Que 7. What is transaction? Draw the state diagram of transaction showing its state. Explain ACID properties of transaction with suitable example.

TRANSACTION SYSTEM

A collection of several operations on the database appears to be a single unit from the point of view of the database user. e.g. a transfer of funds from a checking account to a saving account is a single operation from the customers standpoint. within the database system, however, it consists of several operations.

Collection of operations that form a single logical unit of work are called "transactions".

A database system must ensure proper execution of transactions despite failures - either the entire transaction executes, or none of it does.

TRANSACTION CONCEPT

A transaction is a unit of programs execution that accesses and updates various data items. A transaction is initiated by a user program written in a high-level DML or programming language (e.g. SQL, C++ or JAVA), where it is delimited by statements (or function calls) of the form "begin transaction" and "end transaction". The transaction consist of all operation executed b/w the begin transaction & end transaction.

To ensure integrity of the data, we require that the database system maintain the following properties -

* Atomicity Either all operations of the transaction are reflected properly in the database or none are.
* Consistency
* Isolation
* Durability

These operations are often called the "ACID" properties.

Transactions access data using two operations:

* read(X): which transfers the data item X from database to a local buffer belonging to the transaction.
* write(x) which transfers the data item X from the local buffer of the transaction.

Let T_i be a transaction that transfers $50 from account A to account B. This transaction can be defined as

```
Ti: read(A);
    A:= A-50;
    write(A);
    read(B);
    B:= B+50;
```

ATOMICITY

Either all operations of the transaction are reflected properly in the database, or none are.

The basic idea behind ensuring atomicity is that the database system keeps track (on disk) of the old values of any data on which a transaction performs a write, and, if the transaction does not complete its execution, the database system restores old values to make it appear as though the transaction never executed.

CONSISTENCY

The consistency requirement is that the sum of A and B be unchanged by the execution of the transaction.

If database remain consistent before an execution of the transaction, the database remains consistent after the execution of the transaction.

ISOLATION

Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions T_i and T_j,

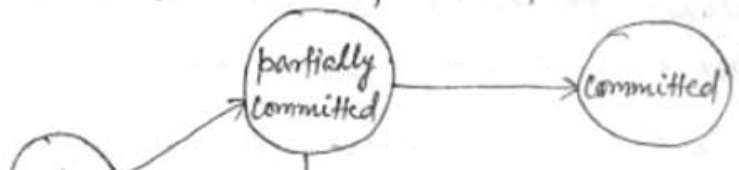
DURABILITY

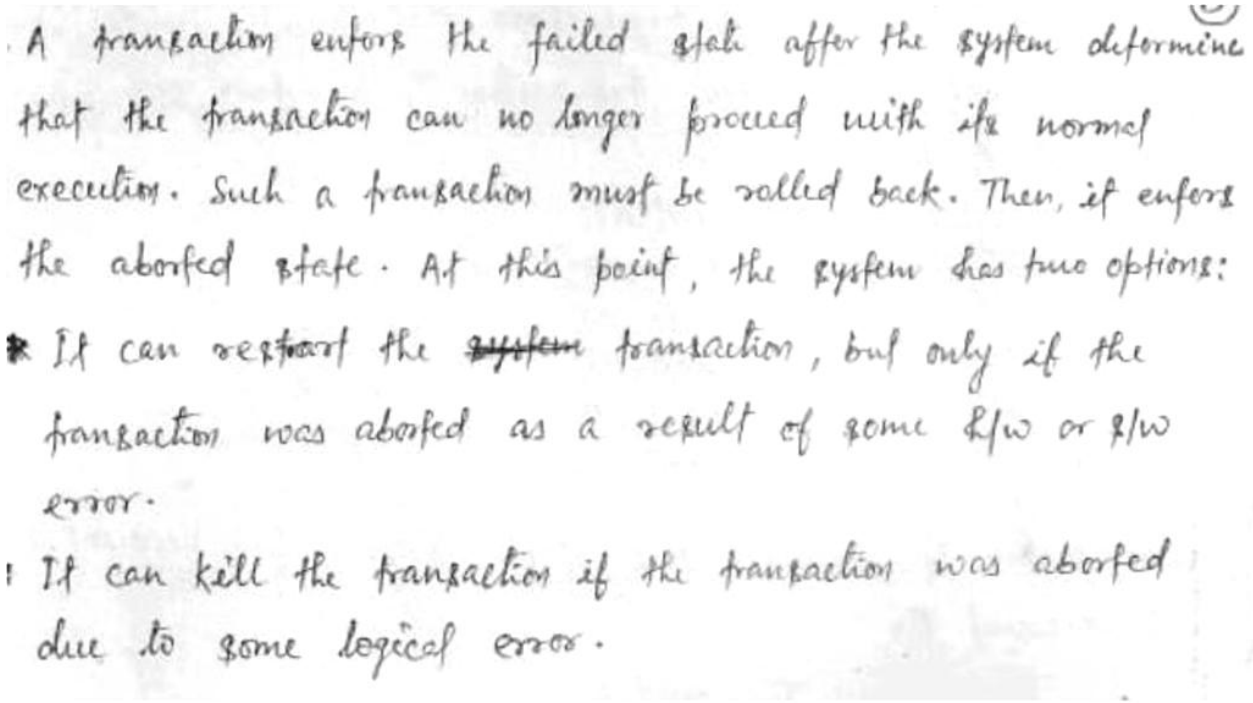
The durability property guarantees that, once a transaction completes successfully, all the updates that it carried out on the database persists, even if there is a system failure after the transaction completes execution.

TRANSACTION STATE

A transaction must be in one of the following states:

* Active, the initial state, the transaction stays in this state while it is executing.
* Partially committed, after the final statement has been executed.
* Failed, after the discovery that normal execution can no longer proceed.
* Aborted, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
* Committed, after successful completion





Que 8. What are schedules? What are differences between conflict serializability and view serializability? Explain with suitable example what are cascadeless and recoverable schedule?

Schedule

A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

Strict Schedule

In Strict schedule, if the write operation of a transaction precedes a conflicting operation (Read or Write operation) of another transaction then the commit or abort operation of such transaction should also precede the conflicting operation of other transaction.

Cascadeless Schedule

In Cascadeless Schedule, if a transaction is going to perform read operation on a value, it has to wait until the transaction who is performing write on that value commits.

Recoverable Schedule

In Recoverable schedule, if a transaction is reading a value which has been updated by some other transaction then this transaction can commit only after the commit of other transaction which is updating value.

SERIALIZABILITY

The database system must control concurrent execution of transactions, to ensure that the database state remains consistent. Before we examine how the database system can carry out this task, we must first understand which schedules will ensure consistency, and which schedule will not.

CONFLICT SERIALIZABILITY

Let us consider a schedule S in which there are two consecutive instructions I_i and I_j, of transactions T_i and T_j, respectively ($i \neq j$). If I_i and I_j refers to different data items, then we can swap I_i and I_j, without affecting the results of any instruction in the schedule. However if I_i and I_j refer to same data item Q, then the order of the two steps may matter. We deal with only read and write. We have four cases to consider —

① I_i = Read(Q), I_j Read(Q), the order of I_i and I_j does not matter, since the same value of Q is read by T_i and T_j.

② I_i = Read(Q), I_j = write(Q). If I_i comes before I_j, then T_i does not read the value of Q that is written by T_j in instruction I_j. If I_j comes before I_i, then T_i reads the value of Q that is written by T_j. Thus, the order of I_i and I_j matters.

③ I_i = write(Q), I_j = read(Q), same as above.

④ I_i = write(Q), I_j = write(Q), since both instructions are write operations, the order of these instructions does not ~~matter~~ affect either T_i or T_j. However, the value obtained by next read(Q) instruction of S is affected.

We say that I_i and I_j conflict if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation. To illustrate the concept of conflicting instructions, consider

schedule 3:

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	

T_1

If we continue the swapping of nonconflicting instructions:

-> swap read(B) of T_1 with read(A) of T_2

-> swap write(B) of T_1 with write(A) of T_2

-> swap write(B) of T_1 with read(A) of T_2

The final result of these swaps is schedule 6 a serial schedule. Thus schedule 3 is equivalent to a serial schedule

If a schedule S can be transformed into a schedule S' by a series of swaps of nonconflicting instructions, we say that S and S' are conflict equivalent.

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

The concept of conflict equivalence leads to the concept of conflict serializability. we say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule. Thus, schedule 3 is conflict serializable, since it is conflict equivalent to the serial schedule 1.

VIEW SERIALIZABILITY

Consider two schedules S and S', where the same set of transactions participate in both schedules. The schedule S and S' are said to be view equivalent if following three conditions are met:

1. Initial Read should be same in both schedule corresponding to T_i.
   For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i must, in schedule S' also read the initial value of Q.

S_1

T_1	T_2
R(A)	
	R(B)

S_2

T_1	T_2
	R(B)
R(A)	

$S_1 = S_2$

2. Updated Read should be same in both schedule.
   For each data item Q, if transaction T_i executes read(Q) in schedule S, and if that value was produced by a write(Q) operation executed by transaction T_j, then the read(Q) operation of transaction T_i must in schedule S' also read the value of Q that was produced by the same write(Q) operation of T_j.

S_1

T_1	T_2	T_3
	W(A)	
R(A)		W(A)

S_2

T_1	T_2	T_3
		W(A)
	W(A)	
R(A)		

$S_1 = S_2$

3. Final Write should be same in both schedule.

The concept of view equivalence leads to the concept of view serializability. We say that a schedule S is view serializable if it is view equivalent to a "serial schedule".

For Example:-

Schedule S

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Schedule U

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

- In 'S' both T_1 & T_2 are reading 'A' and in 'U' T_1 & T_2 are also reading 'A' firstly, thus cond^n (1) is satisfied.
- As T_1 & T_2 in S are firstly reading 'A' then in U T_1 & T_2 are writing 'A' also after reading 'A', thus cond^n (2) is satisfied.
- Finally, in 'S' T_1 & T_2 are writing B whish is also same for 'U', thus cond^n (3) is satisfied.

Hence, serial schedule 'S' & U is view equivalent to 'U', thus they are view serializable.

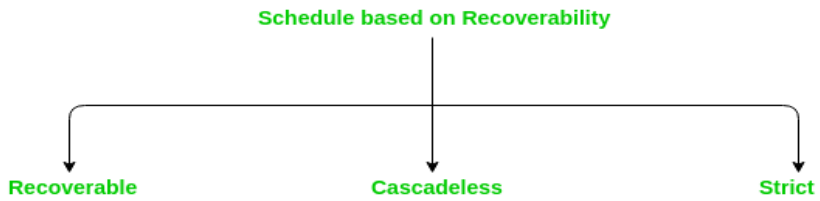
Testing for Serializability of a schedule.

To test for serializability of a schedule, we use precedence graph. A precedence graph (or serialization graph), which is a directed graph $G=(N,E)$ that consists of a set of nodes $N=\{T_1, T_2 \cdots, T_n\}$ and a set of directed edges $E=\{e_1, e_2 \cdots e_m\}$ There is one node in the graph for each transaction T_i in the schedule. Each edge e_i in the graph is of the form $(T_j \rightarrow T_K)$, $1 \le j \le n$, $1 \le K, n$, where T_j is the starting node of e_i and T_K is the ending node of e_i. Such an edge from node T_j to node T_K is created by the algorithm if one of the operations in T_j appears in the schedule before some conflicting operation in T_K.

Algorithm:- Testing conflict serializability of a Schedule S.

1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.
2. For each case in S where T_j executes a read(X) after T_i executes a write(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
3. For each case in S where T_j executes a write(X) after T_i executes a read(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph

Cascadeless in DBMS

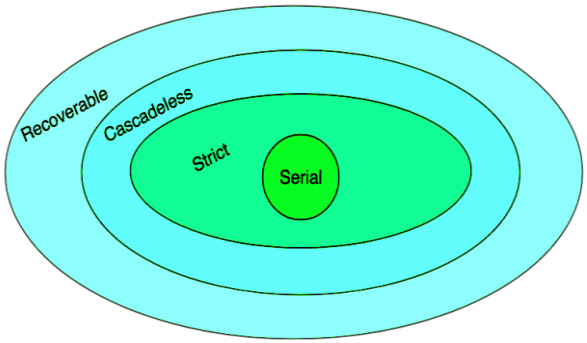


Recoverable schedule:

Transactions must be committed in order. Dirty Read problem and Lost Update problem may occur.

Cascadeless Schedule:

Dirty Read not allowed, means reading the data written by an uncommitted transaction is not allowed. Lost Update problem may occur.



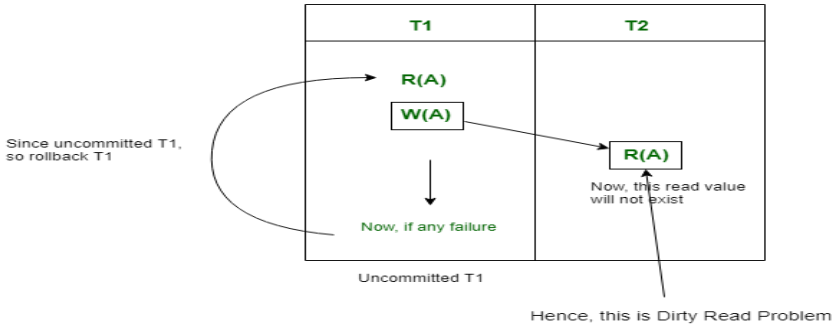
Dirty Read Problem:

When a Transaction reads data from uncommitted write in another transaction, then it is known as *Dirty Read*. If that writing transaction failed, and that written data may updated again. Therefore, this causes *Dirty Read Problem*.

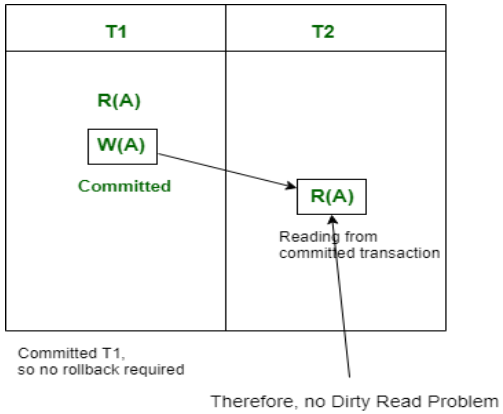
In other words,

It is called as dirty read because there is always a chance that the uncommitted transaction might roll back later. Thus, uncommitted transaction might make other transactions read a value that does not even exist. This leads to inconsistency of the database.

For example, let's say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.



There is no Dirty Read problem, is a transaction is reading from another committed transaction. So, no rollback required.



Cascading Rollback:

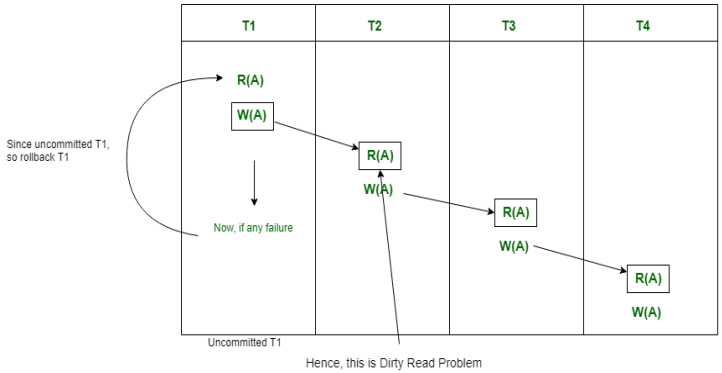
If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Rollback or Cascading Abort or Cascading Schedule. It simply leads to the wastage of CPU time.

These Cascading Rollbacks occur because of Dirty Read problems.

For example, transaction T1 writes uncommitted x that is read by Transaction T2. Transaction T2 writes uncommitted x that is read by Transaction T3.

Suppose at this point T1 fails.

T1 must be rolled back, since T2 is dependent on T1, T2 must be rolled back, and since T3 is dependent on T2, T3 must be rolled back.



Because of T1 rollback, all T2, T3, and T4 should also be rollback (Cascading dirty read problem).

This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks is called Cascading rollback.

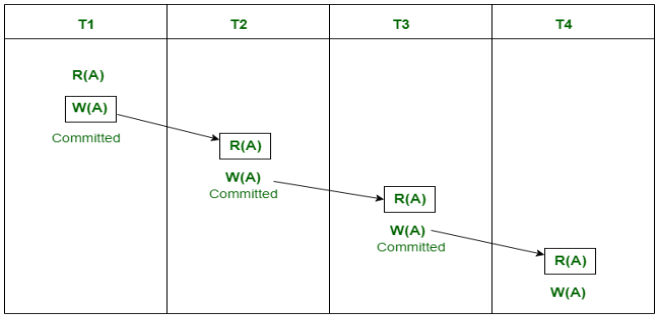
Cascadeless Schedule:

This schedule avoids all possible Dirty Read Problem.

In Cascadeless Schedule, if a transaction is going to perform read operation on a value, it has to wait until the transaction who is performing write on that value commits. That means there must not be Dirty Read. Because Dirty Read Problem can cause Cascading Rollback, which is inefficient.

Cascadeless Schedule avoids cascading aborts/rollbacks. Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Avoids that a single transaction abort leads to a series of transaction rollbacks. A strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.

In other words, if some transaction Tj wants to read value updated or written by some other transaction Ti, then the commit of Tj must read it after the commit of Ti.



No Dirty Read problem

Cascadeless schedule allows only committed read operations. However, it allows uncommitted write operations. Cascadeless Schedules are always recoverable, but all recoverable transactions may not be Cascadeless Schedule.

Que 9. What are distributed database? List the advantages and disadvantages of data replication and data fragmentation. Explain with suitable example. What are differences in replication and fragmentation transparency?

Sol 9. A distributed database is a database that consists of two or more files located in different sites either on the same network or on entirely different networks. Portions of the database are stored in multiple physical locations and processing is distributed among multiple database nodes.

Advantages and Disadvantages of Data Replication in Distributed Database

Data Replication

Data replication is the process where in a relation (a table) or portion of a relation (a fragment of a table) is duplicated and those duplicated copies are stored in multiple sites (servers) to increase the availability of data.

“Replication is the process of copying (duplicating) and maintaining database objects in multiple databases that make up a distributed database system”

Advantages:

1. Increased reliability and availability

We have many copies of same data in several different locations (usually different geographical locations). Hence, failure of any sites (servers) will not affect the transactions.

2. Queries requesting replicated copies of data are always faster

Distributed database ensures the availability of data where it is needed much. In case of replication, this is one step ahead. Yes, the complete table itself loaded locally. Hence, those queries can be answered quickly from the local site where they are initiated.

3. Less communication overhead

When more number of read queries is generated in a site, all of them can be answered locally. Only the queries involving different table or the queries try to write something need to use the communication links to contact other sites.

Disadvantages:

1. More storage space is needed when compared to a centralized system – Replication would mean to duplicate any tables and store them in every site. This need more space in every site.

2. Update operation is costly – If we have more copies of same data loaded in different sites, obviously we need to update all the replicas whenever we would like to change data. Hence, write operation is always costly.

3. Maintaining data integrity is complex – It involves complex procedures to maintain consistent database.

Data Fragmentation

2) Data Fragmentation –

Data Fragmentation is a method in which different relations of a relational database system can be sub-divided and distributed among different network sites. The union of these fragments reconstructs the original relation S.

for ex- Suppose, we have a relation S and it is fragmented it means it is divided into sub-sets (fragments)

$S_1, S_2, S_3 - - - S_n$

There are three types of fragmentation:

1. Horizontal fragmentation - tuples of a relation are divided in subsets
2. Vertical fragmentation attributes are divided into number of subsets
3. Mixed fragmentation - horizontal of a relation, followed by further vertical fragmentation

Soln- We have three types of fragmentation

1) Horizontal Fragmentation —

tuples of relation are divided into number of subsets.

ex- Employ table

Empno	Emp_name	Designation	Salary	Dept_no
1001	Amit	CA	15000	1
1002	Anu	CEO	25000	1
1003	Ravi	MD	20000	2
1004	Rahul	PRO	10000	2

for horizontal fragmentation

$$Employ_1 = \sigma_{Dept\_no = "1"}(Employ)$$

$$Employ_2 = \sigma_{Dept\_no = "2"}(Employ)$$

Thus we have two fragments

Relation Employ1

Emp no	Emp_name	Designation	Salary	Dept_no
1001	Amit	CA	15000	1
1002	Anu	CEO	25000	1

2) Relation Employ 2

Emp_no	Emp_name	Designation	Salary	Dept no
1003	Ravi	MD	20,000	2
1004	Rahul	PRO	10000	2

reconstruction we got by union of relations
Employ = Employ1 ∪ Employ2

Vertical Fragmentation –

attributes are divided into number of subsets.

ex- Emp

Emp_no	Emp_name	Salary	Dept_no	Tuple_id
1001	Amit	15000	1	1
1002	Anu	25000	1	2
1003	Ravi	20000	2	3
1004	Rahul	10000	2	4

here we add tuple_id which show logical

Oracle **distributed database** systems employ a **distributed** processing architecture to function. For **example**, an Oracle server acts as a client when it requests data that another Oracle server manages.

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Features

- Databases in the collection are logically interrelated with each other. Often they represent a single logical database.
- Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.
- The processors in the sites are connected via a network. They do not have any multiprocessor configuration.
- A distributed database is not a loosely connected file system.
- A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Distributed Database Management System

A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

Features

- It is used to create, retrieve, update and delete distributed databases.
- It synchronizes the database periodically and provides access mechanisms by the virtue of which the distribution becomes transparent to the users.
- It ensures that the data modified at any site is universally updated.
- It is used in application areas where large volumes of data are processed and accessed by numerous users simultaneously.
- It is designed for heterogeneous database platforms.
- It maintains confidentiality and data integrity of the databases.

Factors Encouraging DDBMS

The following factors encourage moving over to DDBMS −

- **Distributed Nature of Organizational Units** − Most organizations in the current times are subdivided into multiple units that are physically distributed over the globe. Each unit requires its own set of local data. Thus, the overall database of the organization becomes distributed.
- **Need for Sharing of Data** − The multiple organizational units often need to communicate with each other and share their data and resources. This demands common databases or replicated databases that should be used in a synchronized manner.
- **Support for Both OLTP and OLAP** − Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) work upon diversified systems which may have common data. Distributed database systems aid both these processing by providing synchronized data.
- **Database Recovery** − One of the common techniques used in DDBMS is replication of data across different sites. Replication of data automatically helps in data recovery if database in any site is damaged. Users can access data from other sites while the damaged site is being reconstructed. Thus, database failure may become almost inconspicuous to users.
- **Support for Multiple Application Software** − Most organizations use a variety of application software each with its specific database support. DDBMS provides a uniform functionality for using the same data among different platforms.

Advantages of Distributed Databases

Following are the advantages of distributed databases over centralized databases.

Modular Development − If the system needs to be expanded to new locations or new units, in centralized database systems, the action requires substantial efforts and disruption in the existing functioning. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions.

More Reliable − In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.

Better Response − If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time.

Lower Communication Cost − In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

Adversities of Distributed Databases

Following are some of the adversities associated with distributed databases.

- **Need for complex and expensive software** − DDBMS demands complex and often expensive software to provide data transparency and co-ordination across the several sites.
- **Processing overhead** − Even simple operations may require a large number of communications and additional calculations to provide uniformity in data across the sites.
- **Data integrity** − The need for updating data in multiple sites pose problems of data integrity.
- **Overheads for improper data distribution** − Responsiveness of queries is largely dependent upon proper data distribution. Improper data distribution often leads to very slow response to user requests.

Data Replication is the process of storing data in more than one site or node. It is useful in **improving the availability of data**. It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency. The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others.

Data replication encompasses duplication of transactions on an ongoing basis, so that the **replicate is in a consistently updated state** and synchronized with the source. However in data replication data is available at different locations, but a particular relation has to reside at only one location.

Design Alternatives

The distribution design alternatives for the tables in a DDBMS are as follows −

- Non-replicated and non-fragmented
- Fully replicated
- Partially replicated
- Fragmented
- Mixed

Non-replicated & Non-fragmented

In this design alternative, different tables are placed at different sites. Data is placed so that it is at a close proximity to the site where it is used most. It is most suitable for database systems where the percentage of queries needed to join information in tables placed at different sites is low. If an appropriate distribution strategy is adopted, then this design alternative helps to reduce the communication cost during data processing.

Fully Replicated

In this design alternative, at each site, one copy of all the database tables is stored. Since, each site has its own copy of the entire database, queries are very fast requiring negligible communication cost. On the contrary, the massive redundancy in data requires huge cost during update

operations. Hence, this is suitable for systems where a large number of queries is required to be handled whereas the number of database updates is low.

Partially Replicated

Copies of tables or portions of tables are stored at different sites. The distribution of the tables is done in accordance to the frequency of access. This takes into consideration the fact that the frequency of accessing the tables vary considerably from site to site. The number of copies of the tables (or portions) depends on how frequently the access queries execute and the site which generate the access queries.

Fragmented

In this design, a table is divided into two or more pieces referred to as fragments or partitions, and each fragment can be stored at different sites. This considers the fact that it seldom happens that all data stored in a table is required at a given site. Moreover, fragmentation increases parallelism and provides better disaster recovery. Here, there is only one copy of each fragment in the system, i.e. no redundant data.

The three fragmentation techniques are −

- Vertical fragmentation
- Horizontal fragmentation
- Hybrid fragmentation

Mixed Distribution

This is a combination of fragmentation and partial replications. Here, the tables are initially fragmented in any form (horizontal or vertical), and then these fragments are partially replicated across the different sites according to the frequency of accessing the fragments.

Data Replication

Data replication is the process of storing separate copies of the database at two or more sites. It is a popular fault tolerance technique of distributed databases.

Advantages of Data Replication

- **Reliability** − In case of failure of any site, the database system continues to work since a copy is available at another site(s).
- **Reduction in Network Load** − Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.
- **Quicker Response** − Availability of local copies of data ensures quick query processing and consequently quick response time.
- **Simpler Transactions** − Transactions require less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Disadvantages of Data Replication

- **Increased Storage Requirements** − Maintaining multiple copies of data is associated with increased storage costs. The storage space required is in multiples of the storage required for a centralized system.
- **Increased Cost and Complexity of Data Updating** − Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.
- **Undesirable Application – Database coupling** − If complex update mechanisms are not used, removing data inconsistency requires complex co-ordination at application level. This results in undesirable application – database coupling.

Some commonly used replication techniques are:-

- Snapshot replication
- Near-real-time replication
- Pull replication

Fragmentation

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called fragments. Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical). Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called "reconstructiveness."

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very high.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

Vertical Fragmentation

In vertical fragmentation, the fields or columns of a table are grouped into fragments. In order to maintain reconstructiveness, each fragment should contain the primary key field(s) of the table. Vertical fragmentation can be used to enforce privacy of data.

For example, let us consider that a University database keeps records of all registered students in a Student table having the following schema.

STUDENT

Regd_No	Name	Course	Address	Semester	Fees	Marks

Now, the fees details are maintained in the accounts section. In this case, the designer will fragment the database as follows −

```
CREATE TABLE STD_FEES AS SELECT Regd_No, Fees FROM STUDENT;
```

Horizontal Fragmentation

Horizontal fragmentation groups the tuples of a table in accordance to values of one or more fields. Horizontal fragmentation should also confirm to the rule of reconstructiveness. Each horizontal fragment must have all columns of the original base table.

For example, in the student schema, if the details of all students of Computer Science Course needs to be maintained at the School of Computer Science, then the designer will horizontally fragment the database as follows:-

```
CREATE COMP_STD AS SELECT * FROM STUDENT WHERE COURSE = "Computer Science";
```

Hybrid Fragmentation

In hybrid fragmentation, a combination of horizontal and vertical fragmentation techniques are used. This is the most flexible fragmentation technique since it generates fragments with minimal extraneous information. However, reconstruction of the original table is often an expensive task.

Hybrid fragmentation can be done in two alternative ways −

At first, generate a set of horizontal fragments; then generate vertical fragments from one or more of the horizontal fragments.

At first, generate a set of vertical fragments; then generate horizontal fragments from one or more of the vertical fragments.

Section-C

Que. 10 Describe the major problems associated with concurrent processing with examples. What is the role of locks in avoiding these problems?

Concurrency problems in DBMS Transactions

When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems. These problems are commonly referred to as concurrency problems in database environment. The five concurrency problems that can occur in database are:

(i). Temporary Update Problem

(ii). Incorrect Summary Problem

(iii). Lost Update Problem

(iv). Unrepeatable Read Problem

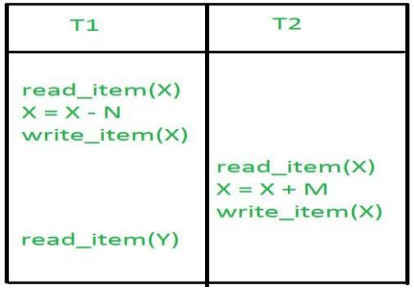
(v). Phantom Read Problem

These are explained as:-

(i). Temporary Update Problem:

Temporary update or dirty read problem occurs when one transaction updates an item and fails. But the updated item is used by another transaction before the item is changed or reverted back to its last value.

Example:

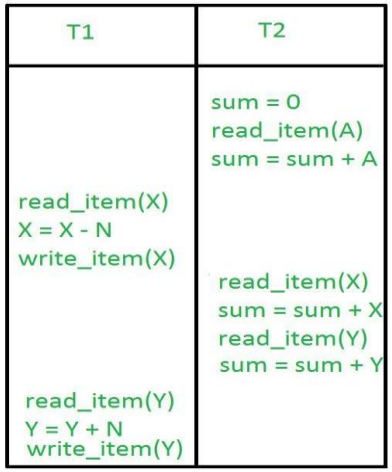


In the above example, if transaction 1 fails for some reason then X will revert back to its previous value. But transaction 2 has already read the incorrect value of X.

(ii). Incorrect Summary Problem:

Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records. The aggregate function may calculate some values before the values have been updated and others after they are updated. In the above example, transaction 2 is calculating the sum of some records while transaction 1 is updating them. Therefore the aggregate function may calculate some values before they have been updated and others after they have been updated.

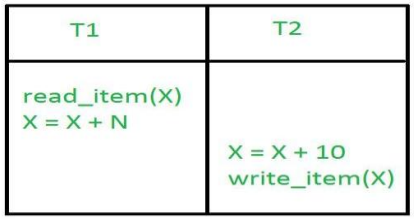
Example:



(iii). Lost Update Problem:

In the lost update problem, update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

Example:

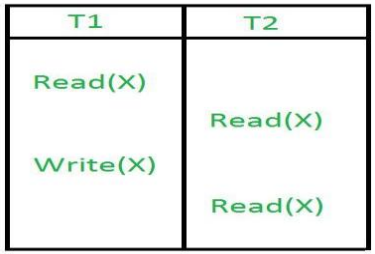


In the above example, transaction 1 changes the value of X but it gets overwritten by the update done by transaction 2 on X. Therefore, the update done by transaction 1 is lost.

(iv). Unrepeatable Read Problem:

The unrepeatable problem occurs when two or more read operations of the same transaction read different values of the same variable.

Example:

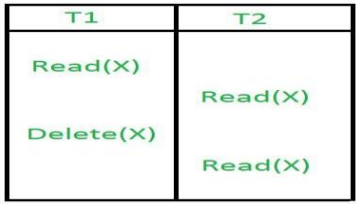


In the above example, once transaction 2 reads the variable X, a write operation in transaction 1 changes the value of the variable X. Thus, when another read operation is performed by transaction 2, it reads the new value of X which was updated by transaction 1.

(v). Phantom Read Problem:

The phantom read problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.

Example:



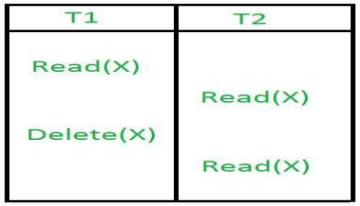
In the above example, once transaction 2 reads the variable X, transaction 1 deletes the variable X without transaction 1's knowledge. Thus, when transaction 2 tries to read X, it is not able to it.

Que 11. Explain the phantom phenomenon. Devise a time stamp based protocol that avoids the phantom phenomenon.

(i) Phantom Read Problem:

The phantom read problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.

Example:



In the above example, once transaction 2 reads the variable X, transaction 1 deletes the variable X without transaction 1's knowledge. Thus, when transaction 2 tries to read X, it is not able to it.

Time stamp based protocol that avoids the phantom:-

Two approaches to dealing with the phantom phenomenon by means of locking. The coarser granularity approach obviously works for timestamps as well. The B+-tree index based approach can be adapted to time stamping by treating index buckets as data items with timestamps associated with them, and requiring that all read accesses use an index. We now show that this simple method works. Suppose a transaction Ti wants to access all tuples with a particular range of search-key values, using a B+- tree index on that search-key. Ti will need to read all the buckets in that index which have key values in that range. It can be seen that any delete or insert of a tuple with a key-value in the same range will need to write one of the index buckets read by Ti. Thus the logical conflict is converted to a conflict on an index bucket, and the phantom phenomenon is avoided.

Que 12. What do you mean by multiple granularities? How it is implemented in transaction system?

Multiple Granularity: It can be defined as hierarchically breaking up the database into blocks which can be locked. The Multiple Granularity protocol enhances concurrency and reduces lock overhead.

In the various Concurrency Control schemes have used different methods and every individual Data item as 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 the certain flaw 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 loss of Concurrency, which was our primary goal in the first place.

Implementation in transaction system:-

It is the size of data item allowed to lock. Now *Multiple Granularity* means hierarchically breaking up the database into blocks which can be locked and can be track what need 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. 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

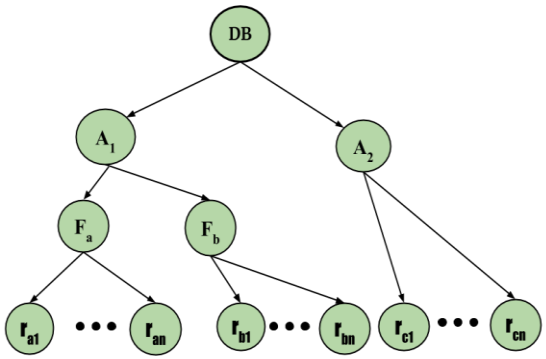


Figure: Multi Granularity tree Hierarchy

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 granularity.

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 granularity:

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 sub-tree 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

Figure – 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 mode. 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 R_{a2} 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 R_{a2} in S mode.

Say transaction T2 modifies record Ra_9 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 R_{a9} 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.

Note that transactions T1, T3 and T4 *can access the database concurrently*. Transaction T2 can execute concurrently with T1, but not with either T3 or T4.

This protocol enhances *concurrency and reduces lock overhead*. Deadlock are still possible in the multiple-granularity protocol, as it is in the two-phase locking protocol. These can be eliminated by using certain deadlock elimination techniques.