Hall Ticket Number:

III/IV B.Tech (Supplementary) DEGREE EXAMINATION

April, 2017 **Common for CSE & IT**

Fifth Semester **Database Management Systems**

Time: Three Hours **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1. Answer all questions (1X12=12 Marks)

a) **Roles of a Database Administrator**
Database administrators (DBAs) use specialized software to store and organize data. The role may include capacity planning, installation, configuration, database design, migration, performance monitoring, security, troubleshooting, as well as backup and data recovery

b) **Schema**
A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

c) **Weak Entity Set**
The entity set which does not have sufficient attributes to form a primary key is called as Weak entity set.

d) **Multi-Valued Attribute**
A multivalued attribute can have more than one value at a time for an attribute. For ex. one person may not have a college degree, another person may have one, and a third person may have two or more degrees; therefore, different people can have different numbers of values for the College_degrees attribute. Such attributes are called multivalued.

e) **Candidate Key**
A candidate key is a column, or set of columns, in a table that can uniquely identify any database record without referring to any other data. Each table may have one or more candidate keys, but one candidate key is special, and it is called the primarykey.

f) **Nested Query**
A Nested query or a Subquery or Inner query is a query within another SQL query and embedded within the WHERE clause. A nested query is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

g) **Record Types**
- Table-based
- Cursor-based records
- User-defined records

h) **Index**
An index is used to speed up the performance of queries. It does this by reducing the number of database data pages that have to be visited/scanned. An index is a copy of selected columns of data from a table that can be searched very efficiently

i) **Functional Dependency**
A functional dependency, denoted by X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y].

j) **Atomicity**
A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.

k) **Exclusive Lock**
When a statement modifies data, its transaction holds anexclusive lock on data that prevents other transactions from accessing the data. This lock remains in place until the transaction

holding the lock issues a commit or rollback.

l) **Shadow Paging.**

Shadow paging is a copy-on-write technique for avoiding in-place updates of pages. Instead, when a page is to be modified, a shadow page is allocated. Since the shadow page has no references (from other pages on disk), it can be modified liberally, without concern for consistency constraints

UNIT I

2. a) **Discuss the main characteristics of the database approach and how it differs from traditional file systems.** **6M**

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

- **Self-describing nature of a database system**
- **Insulation between programs and data, and data abstraction**
- **Support of multiple views of the data**
- **Sharing of data and multiuser transaction processing**

Listing the characteristics ---2 M
Writing any relevant points ---4 M

Self-Describing Nature of a Database System

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

Insulation between Programs and Data and Data Abstraction

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence. In some types of database systems, such as object-oriented and object-relational systems , users can define operations on data as part of the database definitions. An operation is specified in two parts. The interface of an operation includes the operation name and the data types of its arguments. The implementation (or method) of the operation is specified separately and can be changed without affecting the interface. User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.

Support of Multiple Views of the Data

A database typically has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views. For example, one user of the database may be interested only in accessing and printing the transcript of each student; A second user, who is interested only in checking that students have taken all the prerequisites of each course for which they register.

Sharing of Data and Multiuser Transaction Processing

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger. These types of applications are generally called online transaction processing (OLTP) applications. A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.

b) **What is meant by data independence? Explain the difference between physical independence & logical data independence.** 6M

Definition --2M

The three-schema architecture can be used to further explain the concept of data independence, which can be defined as the capacity to change the schema at one level of database system without having to change the schema at the next higher level. We can define two types of data independence:

Types of data Independence--4M

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

(OR)

3. a) **Describe the Classification of Database Management Systems.** **6M**

Writing any relevant points ---6M

Several criteria are normally used to classify DBMSs. The first is the data model on which the DBMS is based. The main data model used in many current commercial DBMSs is the relational data model. The object data model has been implemented in some commercial systems but has not had widespread use.Many legacy applications still run on database systems based on the hierarchical and network data models. Examples of hierarchical DBMSs include IMS (IBM) and some other systems like System 2K (SAS Inc.) and TDMS. IMS is still used at governmental and industrial installations, including hospitals and banks, although many of its users have converted to relational systems. The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called object-relational DBMSs.We can categorize DBMSs based on the data model: relational, object, object-relational, hierarchical, network, and other.

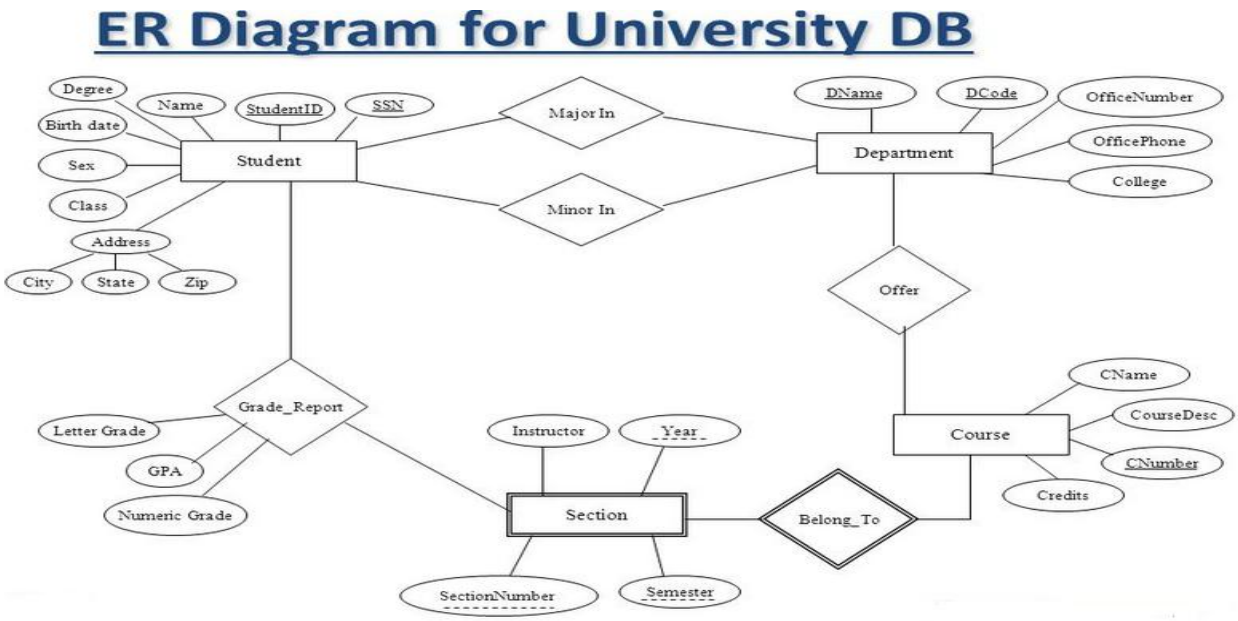
More recently, some experimental DBMSs are based on the XML (eXtended Markup Language) model, which is a tree-structured (hierarchical) data model. These have been called native XML DBMSs. Several commercial relational DBMSs have added XML interfaces and storage to their products.

The second criterion used to classify DBMSs is the number of users supported by the system. Single-user systems support only one user at a time and are mostly used with PCs. Multiuser systems, which include the majority of DBMSs, support concurrent multiple users.

The third criterion is the number of sites over which the database is distributed. A DBMS is centralized if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site. A distributed DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. Homogeneous DDBMSs use the same DBMS software at all the sites, whereas heterogeneous DDBMSs can use different DBMS software at each site.

b) **Construct an E-R diagram for university registrar's office. The office maintains data about each class, including the instructor, the enrolment and the time and place of the class meetings. For each student class pair, a grade is recorded. Determine the entities and relationships that exist between the entities..** 6M

- An entity-relationship diagram (ERD) is a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities. An ERD is a conceptual and representational model of data used to represent the entity framework infrastructure.



UNIT II

4. a) **Differentiate between entity and referential integrity constraints.** 6M

Entity Integrity Constraint [Explanation--2M, Example--1M]

The Integrity Rule 1 is also called Entity Integrity Rule or Constraint. This rule states that no attribute of primary key will contain a null value. If a relation have a null value in the primary key attribute, then uniqueness property of the primary key cannot be maintained. Consider the example below

[Any relevant example can be considered]

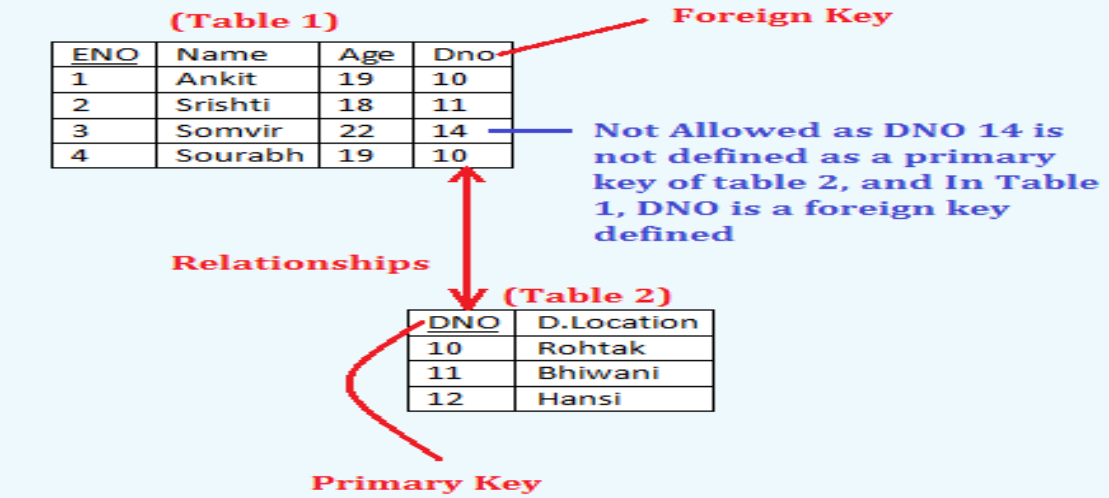
SID	Name	Class (semester)	Age
8001	Ankit	1st	19
8002	Srishti	2nd	18
8003	Somvir	4th	22
	Sourabh	6th	19

Not allowed as primary key cannot contain a NULL value

[Any relevant example can be considered]

Referential Integrity Constraint [Explanation--2M, Example--1M]

The integrity Rule 2 is also called the Referential Integrity Constraints. This rule states that if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2. For example,



b) **Consider the following database relations. Write SQL statements given below:** **6M**

S (S#, SNAME, SCITY) P (P#, PNAME, PCITY)

J (J#, JNAME, JCITY) SPJ (S#, P#, J#, QTY)

Each Query--- 2M

i. Get J# values for projects using one part available for supplier?

ii. Get P# values for part supplied to any project in London by a London supplier.

iii. Get JNAME for projects supplied by at least one supplier not in the same city.

(OR)

5. a) **What is a view? Discuss the problems that may arise when one attempts to update a view.** 4M

Definition--2M

A view in SQL terminology is a single table that is derived from other tables.These other tables can be base tables or previously defined views. A view does not necessarily exist in physical form; it is considered to be a virtual table, in contrast to base tables, whose tuples are always physically stored in the database.

Problems when one attempts to update a view--2M

Updating of views is complicated and can be ambiguous. In general, an update on a view defined on a single table without any aggregate functions can be mapped to an update on the underlying base table under certain conditions. For a view involving joins, an update operation may be mapped to update operations on the underlying base relations in multiple ways. Hence, it is often not possible for the DBMS to determine which of the updates is intended. To illustrate potential problems with updating a view defined on multiple tables, consider the WORKS_ON1

view, and suppose that we issue the command to update the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'. This view update is shown in below:
UPDATEWORKS_ON1 SET Pname = 'ProductY' WHERE Lname='Smith' AND Fname='John' AND Pname='ProductX'.
This query can be mapped into several updates on the base relations to give the desired update effect on the view. In addition, some of these updates will create additional side effects that affect the result of other queries.

b) **Briefly explain the following with examples** 8M
   i. **RENAME**
   ii. **CROSS JOIN**

Any relevant example can be considered--2M
Explanation --2M

RENAME operation—which can rename either the relation name or the attribute names, or both—as a unary operator. The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:

$$\rho_{S(B1, B2, \ldots, Bn)}(R) \quad \text{or} \quad \rho_S(R) \quad \text{or} \quad \rho_{(B1, B2, \ldots, Bn)}(R)$$

where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and $B_1, B_2, \ldots, B_n$ are the new attribute names. The first expression renames both the relation and its attributes, the second renames the relation only, and the third renames the attributes only. If the attributes of R are $(A_1, A_2, \ldots, A_n)$ in that order, then each A_i is renamed as B_i.

In SQL, a single query typically represents a complex relational algebra expression. Renaming in SQL is accomplished by aliasing using **AS**, as in the following example:

```
SELECT   E.Fname AS First_name, E.Lname AS Last_name, E.Salary AS Salary
FROM     EMPLOYEE AS E
WHERE    E.Dno=5,
```

CROSS JOIN

Any relevant example can be considered--2M
Explanation --2M

CARTESIAN PRODUCT operation—also known as **CROSS PRODUCT** or **CROSS JOIN**–which is denoted by ×. This is also a binary set operation, but the relations on which it is applied do *not* have to be union compatible. In its binary form, this set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set). In general, the result of $R(A_1, A_2, \ldots, A_n) \times S(B_1, B_2, \ldots, B_m)$ is a relation Q with degree $n + m$ attributes $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$, in that order. The resulting relation Q has one tuple for each combination of tuples—one from R and one from S. Hence, if R has n_R tuples (denoted as $|R|| = n_R$), and S has n_S tuples, then $R \times S$ will have $n_R * n_S$ tuples.

The n-ary CARTESIAN PRODUCT operation is an extension of the above concept, which produces new tuples by concatenating all possible combinations of tuples from n underlying relations.

In general, the CARTESIAN PRODUCT operation applied by itself is generally meaningless. It is mostly useful when followed by a selection that matches values of attributes coming from the component relations. For example, suppose that we want to retrieve a list of names of each female employee's dependents. We can do this as follows:

$$\text{FEMALE\_EMPS} \leftarrow \sigma_{\text{Sex='F'}}(\text{EMPLOYEE})$$
$$\text{EMPNAMES} \leftarrow \pi_{\text{Fname, Lname, Ssn}}(\text{FEMALE\_EMPS})$$
$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$$
$$\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{\text{Ssn=Essn}}(\text{EMP\_DEPENDENTS})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Dependent\_name}}(\text{ACTUAL\_DEPENDENTS})$$

UNIT III

6. a) **Briefly explain different types of single level ordered Indexes.** 6M

Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

Indexing is defined based on its indexing attributes. Indexing can be of the following types.

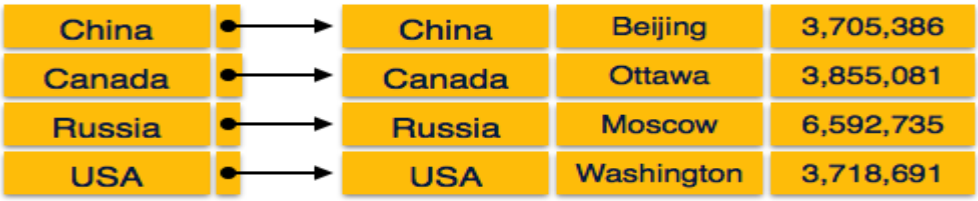
- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a **key field**. The key field is generally the primary key of the relation.
- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Ordered Indexing is of two types −

- Dense Index
- Sparse Index

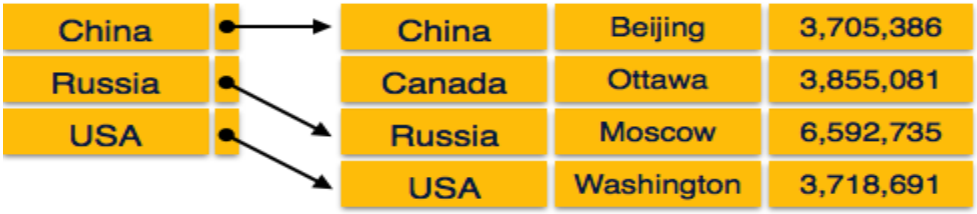
Dense Index

In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.



Sparse Index

In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



b) **Compare BCNF and 3NF with an example.** 6M

Any Relevant Example---2M
Explanation----------4M

Both 3NF and BCNF are normal forms that are used in relational databases to minimize redundancies in tables. In a table that is in the BCNF normal form, for every non-trivial functional dependency of the form A → B, A is a super-key whereas, a table that complies with 3NF should be in the 2NF, and every non-prime attribute should directly depend on every candidate key of that table. BCNF is considered as a stronger normal form than the 3NF and it was developed to capture some of the anomalies that could not be captured by 3NF. Obtaining a

table that complies with the BCNF form will require decomposing a table that is in the 3NF. This decomposition will result in additional join operations (or Cartesian products) when executing queries. This will increase the computational time. On the other hand, the tables that comply with BCNF would have fewer redundancies than tables that only comply with 3NF. Furthermore, most of the time, it is possible to obtain a table that comply with 3NF without hindering dependency preservation and lossless joining. But this is not always possible with BCNF.

The difference between 3NF and BCNF is that for a functionaldependency A → B, 3NF allows this dependency in a relation if B is a primary-key attribute and A is not a candidate key whereas BCNF insists that for this dependency to remain in a relation, A must be a candidate key.

ClientNo	interviewDate	interviewTime	staffNo	roomNo
CR76	13-May-02	10.30	SG5	G101
CR76	13-May-02	12.00	SG5	G101
CR74	13-May-02	12.00	SG37	G102
CR56	1-Jul-02	10.30	SG5	G102

FD1 clientNo, interviewDate → interviewTime, staffNo, roomNo **(Primary Key)**
FD2 staffNo, interviewDate, interviewTime→ clientNo **(Candidate key)**
FD3 roomNo, interviewDate, interviewTime → clientNo, staffNo **(Candidate key)**
FD4 staffNo, interviewDate → roomNo **(not a candidate key)**

As a consequence the ClientInterview relation may suffer from update anmalies.

For example, two tuples have to be updated if the roomNo need be changed for staffNo SG5 on the 13-May-02.

To transform the ClientInterview relation to BCNF, we must remove the violating functional dependency by creating two new relations called Interview and StaffRoom as shown below,

Interview (clientNo, interviewDate, interviewTime, staffNo)
StaffRoom(staffNo, interviewDate, roomNo)

Interview

ClientNo	interviewDate	interviewTime	staffNo
CR76	13-May-02	10.30	SG5
CR76	13-May-02	12.00	SG5
CR74	13-May-02	12.00	SG37
CR56	1-Jul-02	10.30	SG5

Staff Room

staffNo	interviewDate	roomNo
SG5	13-May-02	G101
SG37	13-May-02	G102
SG5	1-Jul-02	G102

(OR)

7. a) **What is normalization? Why it is required?** 4M

Definition --2M

Normalization is the process of efficiently organizing data in a database with two goals in mind
First goal: eliminate redundant data for example, storing the same data in more than one table
Second Goal: ensure data dependencies make sense for example, only storing related data in a table .

Advantages of Normal forms **2M**

Less storage space
Quicker updates
Less data inconsistency
Clearer data relationships
Easier to add data

b) **Define 4NF. Why is it useful?** 8M

Definition--4M
Explanation--4M

A multivalued dependency $X \rightarrow\rightarrow Y$ specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples $t1$ and $t2$ exist in r such that $t1[X] = t2[X]$, then two tuples $t3$ and $t4$ should also exist in r with the following properties,15 where we use Z to denote $(R - (X \cup Y))$:16

- $t3[X] = t4[X] = t1[X] = t2[X]$.
- $t3[Y] = t1[Y]$ and $t4[Y] = t2[Y]$.
- $t3[Z] = t2[Z]$ and $t4[Z] = t1[Z]$.

Whenever $X \rightarrow\rightarrow Y$ holds, we say that X **multidetermines** Y. Because of the symmetry in the definition, whenever $X \rightarrow\rightarrow Y$ holds in R, so does $X \rightarrow\rightarrow Z$. Hence, $X \rightarrow\rightarrow Y$ implies $X \rightarrow\rightarrow Z$, and therefore it is sometimes written as $X \rightarrow\rightarrow Y|Z$.
Dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B and a set of values for C. However, set of values for B and C are independent of each other
MVD between attributes A, B, and C in a relation using the following notation:

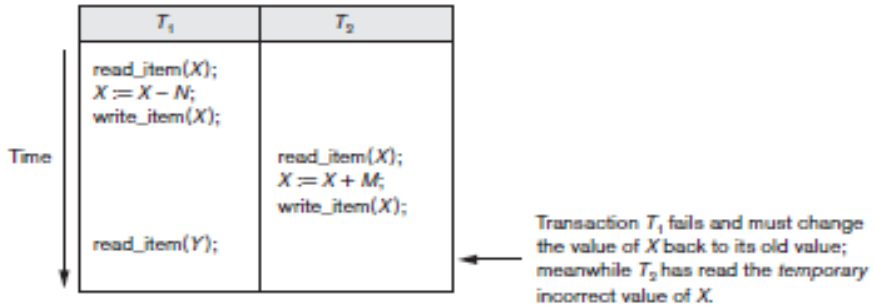
A →> B
A →> C

MVD can be further defined as being trivial or nontrivial.
MVD A →> B in relation R is defined as being trivial if (a) B is a subset of A *or* (b) $A \cup B = R$.
MVD is defined as being nontrivial if neither (a) nor (b) are satisfied.
Trivial MVD does not specify a constraint on a relation, while a nontrivial MVD does specify a constraint.

UNIT IV

8. a) **Describe the Dirty-Read problem**. 4M

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. 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.



b) **What is the two-phase locking protocol? What are the variations of it? How does it guarantee serializability?** 8M

2PL Definition---2M

Two-phase locking techniques---6M

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: an expanding or growing (first) phase, during which new locks on items can be acquired but none can be released; and a shrinking (second) phase, during which existing locks can be released but no new locks can be acquired.

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

- Lock based protocols
- Time stamp based protocols

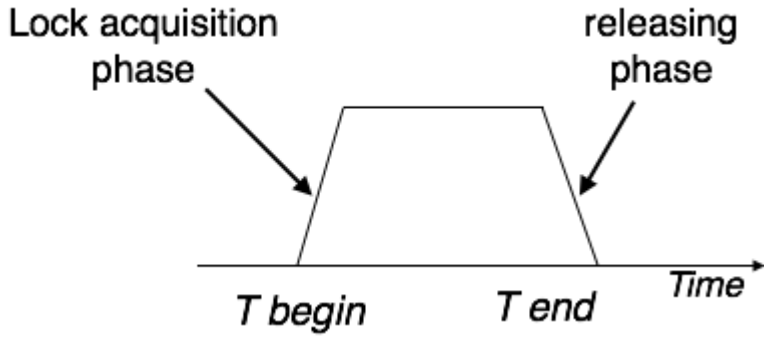
Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds −

- **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive** − This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Two-Phase Locking 2PL

- This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

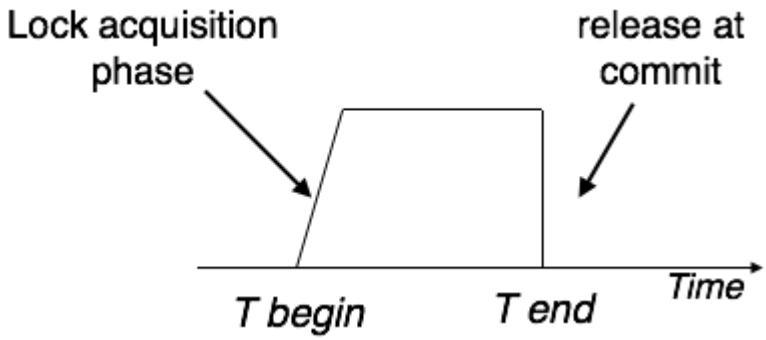


- Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is **shrinking,** where the locks held by the transaction are being released.

- To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking

- The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

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.

(OR)

9. a) **Explain about Recovery Techniques based on Deferred Update.** 6M

Recovery Techniques Based on Deferred Update **[Explanation--6M]**

These techniques defer or postpone any actual updates to the database until the transaction reaches it commit point. During transaction execution, the updates are written to the log file. After the transaction reaches it commit point, the log file is force-written to disk, then the updates are recorded in the database. If the transaction fails before reaching its commit point, there is no need to undo any operations because the transaction has not affected the database on disk in any way.
A typical deferred update protocol uses the following procedure:
A transaction cannot change the database on disk until it reaches its commit point. A transaction does not reach its commit point until all its update operations are recorded in the log file and the log file is force-written to disk. Recovery techniques based on deferred update are therefore known as NO UNDO/REDO techniques. REDO is needed in case the system fails after a

transaction commits but before all its changes are recorded on disk. In this case, the transaction operations are redone from the log file.

Recovery Using Deferred Update in a Single-User Environment
RDU_S (**R**ecovery using **D**eferred **U**pdate in a **S**ingle-User environment) uses A REDO procedure as follows:

PROCEDURE RDU_S:

- Use two lists of transactions: the committed transactions since the last checkpoint, and the active transactions (at most one because the system is single-user).
- Apply the following REDO operation to all the WRITE_ITEM operations of the committed transactions and restart the active transactions:

REDO(WRITE_OP): Redoing a write_item operation WRITE_OP consists of examining its log entry [write_item,T,X,old_value,new_value] and setting the value of item X in the database to its new_value, which is the after image (AFIM).

Deferred Update with Concurrent Execution in a Multiuser Environment
RDU_M(**R**ecovery Using **D**eferred **U**pdate in a **M**ultiuser environment) algorithm is as follows where the REDO procedure is as defined above:

PROCEDURE RDU_M:

- Use two lists of truncations:

  - T: committed transactions since the last checkpoint (commit list).
  - T' : active transactions (active list).

- REDO all the WRITE operations of the committed transactions from the log file, in the order in which they were written in the log.
- The transactions that are active and did not commit are cancelled and must be resubmitted.

The previous algorithm can be made more efficient by noting that if a database item X has been updated more than once by committed transactions since the last checkpoint, it is only necessary to REDO the last update of X from the log file during recovery as the other updates would be overwritten by the last update anyway.
To implement this, start from the end of the log; then whenever an item is redone, it is added to the list of redone items. Before a REDO is applied to an item, the list is checked; if the item appears on the list, it is not redone again, since its most updated value has already been recovered.

b) **Discuss about Granting and Revoking Privileges in detail.** 6M

Any Relevant explanation can be considered----------6M

SQL statements can be issued to perform granting and revoking privileges through any of the Oracle client tools, such as SQL*Plus.

1) **Granting Privileges on Tables**

You can grant users various privileges to tables. These privileges can be any combination of select, insert, update, delete, and others.

The syntax for granting privileges on a table is:

GRANT PRIVILEGES ON OBJECT TO USER;

For example, if you want to grant select, insert, update, and delete privileges on a table called SUPPLIERS to a user name SMITHJ, you would execute the following statement:

GRANT SELECT, INSERT, UPDATE, DELETE ON SUPPLIERS TO SMITHJ;

2) Revoking Privileges on Tables

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can execute a revoke command. You can revoke any combination of select, insert, update, delete, and others.

The syntax for revoking privileges on a table is:

REVOKE PRIVILEGES ON OBJECT FROM USER;

For example, if you want to revoke delete privileges on the table SUPPLIERS from the user SMITHJ, you would execute the following statement:

REVOKE DELETE ON SUPPLIERS FROM SMITHJ.

Scheme prepared by **Signature of HOD, IT Dept**

Paper Evaluators:

Sno	**Name of the college**	**Name of the examiner**	**Signature**