Attempt all questions.

1. Answer the following questions in short:

a. Differentiate between logical data independence and physical data independence.

Solution: The ability to modify schema definition in on level without affecting schema definition in the next higher level is called data independence. There are two types of data independences and their differences are as below:

Physical data independence is the ability to modify the physical schema without causing application programs to be rewritten. Modifications at the physical level are occasionally necessary to improve performance. It means we change the physical storage/level without affecting the conceptual or external view of the data. The new changes are absorbed by mapping techniques.

Logical data independence is the ability to modify the logical schema without causing application program to be rewritten. Modifications at the logical level are necessary whenever the logical structure of the database is altered (for example, when money-market accounts are added to banking system).The correspondence between the conceptual view and the stored database is called **conceptual/internal mapping**. It specifies how conceptual records and relations (files) are represented at the internal level. If a change is made to the storage structure definition, then the conceptual/internal mapping must be changed accordingly, so that the conceptual schema can remain invariant. It is the responsibility of the DBA to manage such changes.

b. Three-schema architectures.

Solution: Data abstraction is the process of easy interface to users by hiding underlying complexities of data management from users. Data abstraction is provided in database management systems by using three-level schema architecture.

Physical level

It is the lowest level of abstractions and describes **how** the data are actually stored on disk and some of the access mechanisms commonly used for retrieving this data. While designing this layer, the main objective is to optimize performance by minimizing the number of disk accesses during the various database operations. **DBMS developer** is the person who deals with this level. Database administrator may be aware of certain details of the physical organization of the data.

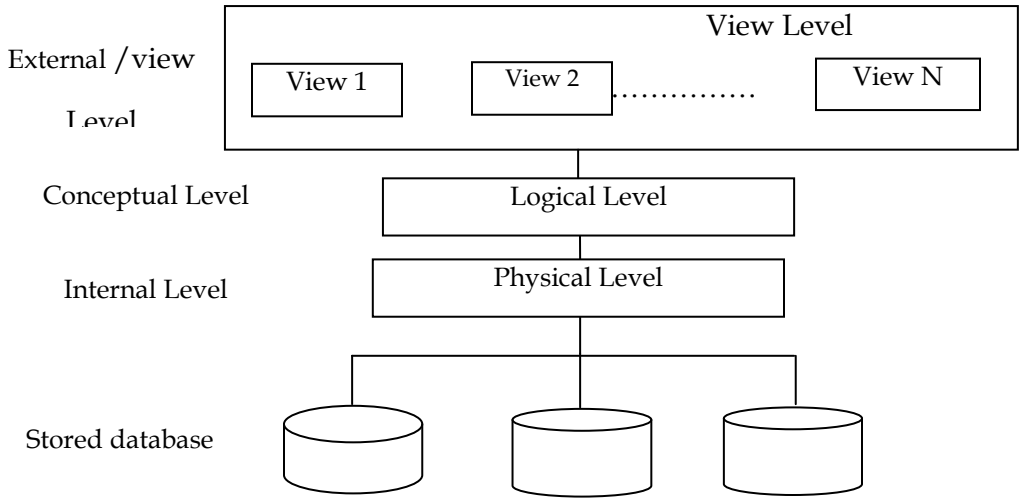


Fig: *Three levels of data abstraction*

Logical level

It is the next higher level of data abstraction which describes **what** data are stored in the database, and what relationships exist among those data. It is also known as conceptual level at the schema at this level is called **logical schema** (conceptual schema). Logical level describes the stored data in terms of the data model of the DBMS. In a relational DBMS, the conceptual schema is described by using **relations (tables)** that are stored in the database. Programmers and database administrator works at this level of abstraction. Database users do not need to have knowledge of this level.

View level

It is the highest level of abstraction and describes only a **part of the database** and hides some information from the user. At view level, computer users see a set of application programs that hide details of data types. Similarly at the view level **several views** of the database are defined and database user see only these views. Schema at this level is called **external schema** (subschema). A view is conceptually a relation, but the records in a **view are not stored** in the DBMS. Rather, they are computed using a definition for the view, from relations stored in the DBMS. Views also plays vital role to provide **security** mechanism to prevent users from accessing certain parts of the database (that is views can also hide information (such as an employee's salary) for security purposes.)

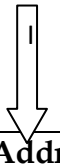
c. Differentiate between database schema and a database state.

Solution: When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the empty state with no data. We get the initial state of the database when the database is first populated or

loaded with the initial data. From then on, every time an update operation is applied to the database, we get another database state. The database schema is set unless you change how the database is structured. The database state will change whenever new data is entered into the database and the database is updated.
The overall structure of the database is called database schema. In the relational model, the schema for a relation specifies its name, the name of each field (or attribute or column), and the type of each field. For example, employee information in a company database may be stored in a relation with the following schema:
Employee (Eid: string, Ename: string, Address: string, Salary: integer, age: integer)

The collection of information stored in the database at a particular moment is called an instance of the database. It is the actual content of the database at a particular point in time. Database instance changes frequently with every insertion, deletion and update operations performed in data stored in a database. For example, one instance of database for above relation schema can be given as below:

Eid	Ename	Address	Salary	Age
E01	Ayan	Kathmandu	35000	25
E02	Bishnu	Pokhara	27000	28
E03	Dina	Dharan	25000	26
E04	Bhanu	Ghorahi	32000	37
E05	Renuka	Dhangadhi	28000	27



Insertion of data causes change in instance

Eid	Ename	Address	Salary	Age
E01	Ayan	Kathmandu	35000	25
E02	Bishnu	Pokhara	27000	28
E03	Dina	Dharan	25000	26
E04	Bhanu	Ghorahi	32000	37
E05	Renuka	Dhangadhi	28000	27
E06	Elish	Mahendranagar	30000	25

Figure: Change in instance due to insertion of new data

d. Different type of data attributes.
Solution: Several types of attributes occur in the ER model: simple versus composite, single-valued versus multi-valued, and stored versus derived.
1. Simple and composite attribute
The attributes that cannot be divided into subparts are called **simple (atomic) attributes**. For example roll-number attribute of a student cannot be further divided into sub parts thus roll-number attribute of a student entity acts as a simple

attribute. The attributes that can be divided into subparts are called **composite attributes**. For example name attribute of a particular student can be further vided into subparts first-name, middle-name, and last-name thus name attribute acts as a composite.

Composite attributes are useful to model situations in which a user sometimes refers to the composite attribute as a unit but at other times refers specifically to its components. If the composite attribute is referenced only as a whole, there is no need to subdivide it into component attributes. Examples of simple and composite attributes are shown below:

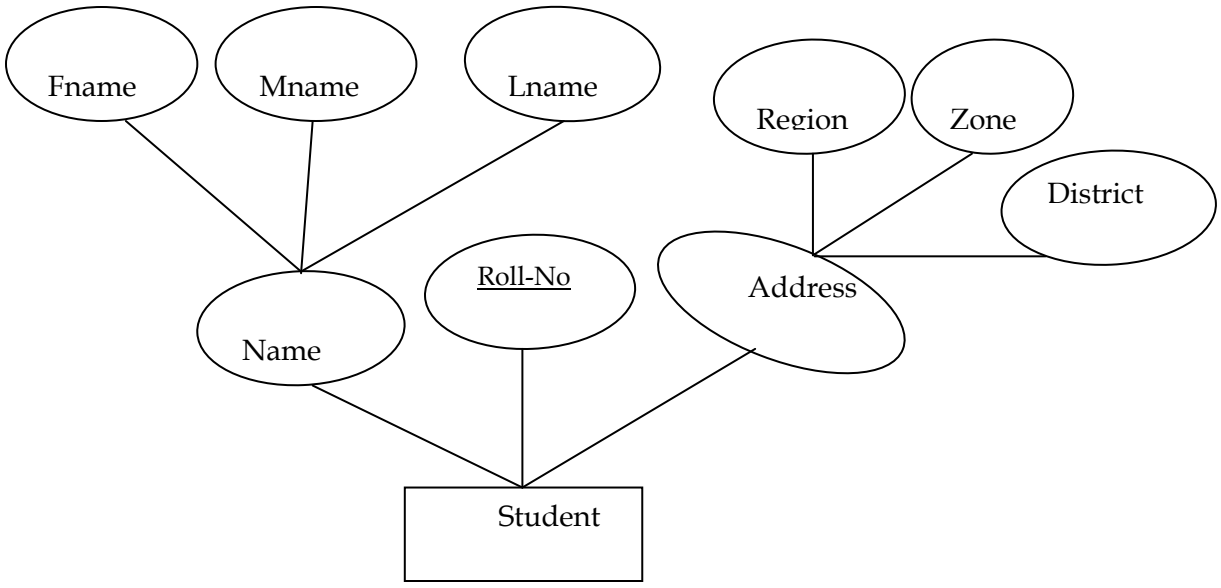
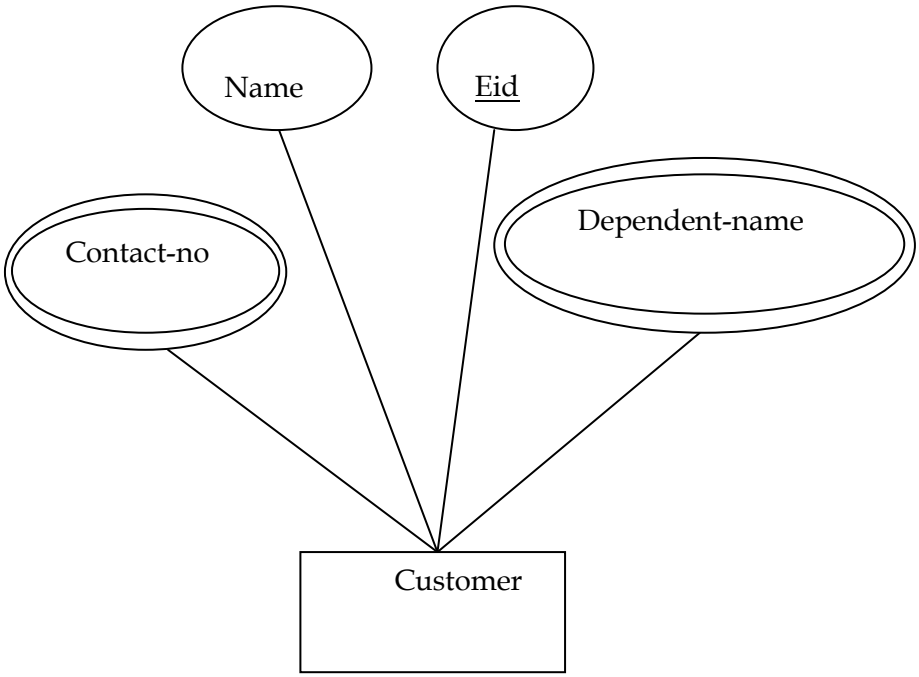


Figure: Simple and composite attributes of Student entity

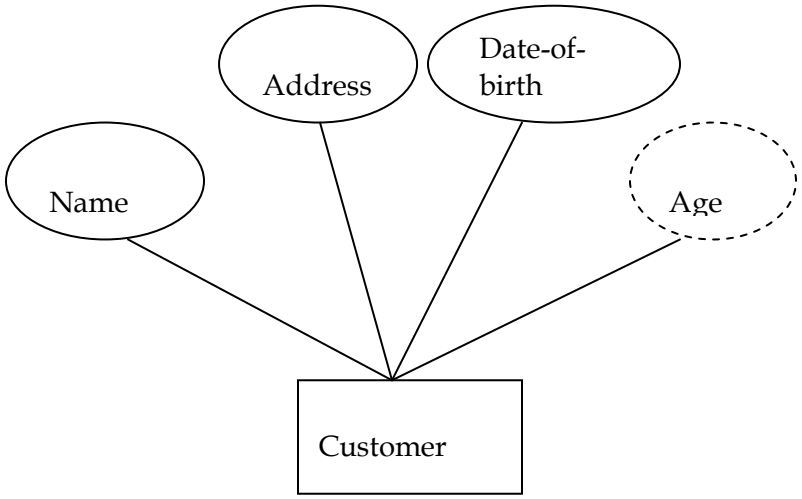
2. Single-valued and multi-valued attributes

The attributes which can have only one value are called **single-valued attributes**. For example, **age** of an employee is single-valued attribute because a person cannot have two value of age. An attribute that can have more than one values are called multi-valued attributes. An employee can have more than one contact numbers and also may have more than one dependent therefore **contact-number** and **dependent-name** are examples of multi-valued attributes. Multi-valued attributes are represented by **double oval** in ER-diagram.



3. Stored versus Derived Attributes

An attribute whose value needs not to be stored rather it can be computed from other attributes or entities is called **derived attribute**. And the attribute whose value is stored in database is called **stored attribute**. For a particular person entity, the value of age can be determined from the current (today's) date and the value of that person's Birth date. The Age attribute is hence called a **derived attribute** and is said to be derivable from the Birth date attribute, which is called stored attribute. Some attribute values can be derived from related entities; for example, an attribute Number-of-employees of a 'Department' entity can be derived by counting the number of employees related to (working for) that department. Derived attributes are represented by dotted oval in ER diagram.

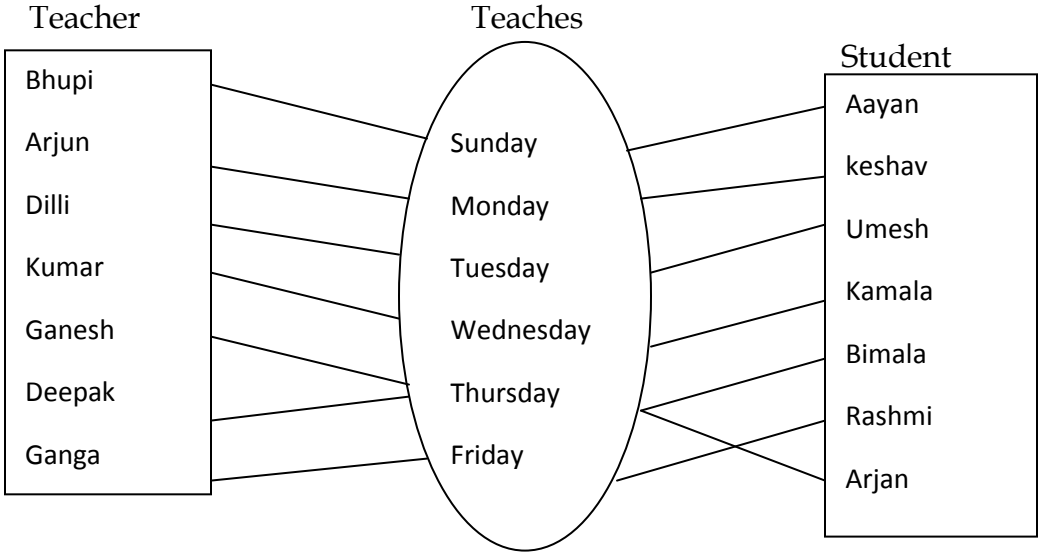


e. The difference among a relationship instance, a relationship type, and relationship set.

Solution: The relationship among Particular two or more entities of a entity set is called relationship instance.

The relationship among two or more entity sets is called relationship set.

A relationship type is an abstraction of a relationship i.e. a set of relationships instances sharing common attributes.

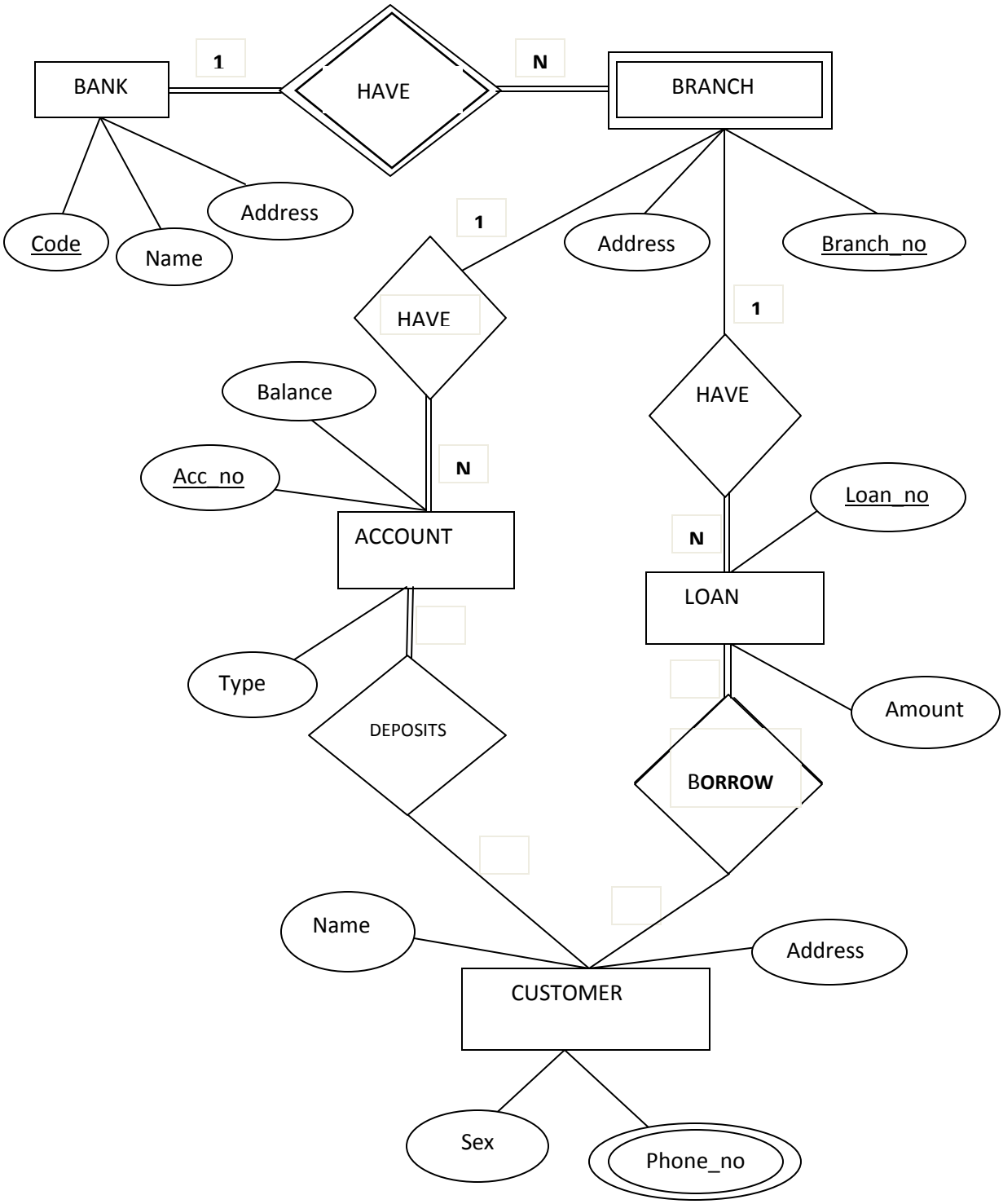


In the above fig teacher teaches student is called relationship set. If we take a particular instance i.e. Bhupi teaches at Sunday to Aayan is called relationship instance. And finally if each teacher uses same attributes and also each student uses same attributes then association among them is called relationship type. The relationship type may be one of the below:

one to one
one to many
many to one
many to many

2. a. Draw an ER diagram for database showing Bank. Each Bank can have multiple branches, and each branch can have multiple accounts and loans.

Solution:



2. b. In what sense does a relational calculus differ from relational algebra, and in what sense are they similar?

Solution: The difference between relational calculus and relational algebra are listed below:

1. Relational algebra operations manipulate some relations and provide some expression in the form of queries where as relational calculus are formed queries on the basis of pairs of expressions.

2. RA have operator like join, union, intersection, division, difference, projection, selection etc. whereas RC has tuples and domain oriented expressions.
3. RA is procedural language where as RC is non procedural query system.
4. There is modification which is easy in queries in RA than the RC.
5. Relational algebra is easy to manipulate and understand than RC.
6. RA queries are more powerful than the RC.

The similarity between relational calculus and relational algebra are listed below:

1. Both relational algebra and relational calculus are formal languages associated with relational model that are used to specify the basic retrieval requests.
2. Expressive power of RA and RC are equivalent. This means any query that could be expressed in RA could be expressed by formula in RC.

3. **Assume a database about Company.**
Employee (ss#, name)
Company (cname, address)
Works (ss#, cname)
Superviser (superviser_ss#, employee_ss#)

a) Write relational algebra and SQL queries for each of the following cases.

i. Find the names of all supervisors that work in companies whose address equals 'pokhara'.
Note: ss= Social security number

```
SELECT name
FROM Employee e, Supervise s, Works w, Company c
WHERE C.address='Pokhara' AND e.ss#=s.supervisor_ss# AND s.supervisor_ss# = w.ss# AND w.cname=c.cname;
```

ii. Find the name of all the companies who have more than 4 supervisors.

```
SELECT cname, COUNT (supervisor_ss#) AS nos
FROM Company c, Works w, Superviser s
WHERE C.cname=w.cname AND w.ss#=s.Supervisor_ss#
GROUP BY cname
HAVING nos>4;
```

iii. Find the name of supervisor who has the largest number of employees.

```
SELECT name, MAX (noe)
FROM (SELECT name, supervisor_ss#, COUNT (employee_ss#) AS noe
        FROM Employee e, Supervise s
            WHERE e.ss#=s.supervisor_ss#
                GROUP BY supervisor_ss#);
```

b) What is a view in SQL and how it is defined? Explain how views are typically implemented.

Solution: A database view is a logical table. It does not physically store data like tables but represent data stored in underlying tables in different formats. A view does not require desk space and we can use view in most places where a table can be used. Since the views are derived from other tables thus when the data in its source tables are updated, the view reflects the updates as well. They also can be used by DBA to enforce database security.

Advantages of Views:

- Database security: view allows users to access only those sections of database that directly concerns them.
- View provides data independence.
- Easier querying
- Shielding from change
- Views provide group of users to access the data according to their criteria.
- Vies allow the same data to be seen by different users in different ways at the same time.

When the column of a view is directly derived from the column of a base table, that column inherits any constraints that apply to the column of the base table. For example, if a view includes a foreign key of its base table, INSERT and UPDATE operations using that view are subject to the same referential constraints as the base table.

Syntax for creating view is:

CREATE VIEW <view name> AS <query expression>

Where, <query expression> is any legal query expression.

Example: Following view contains the id, name, level, age and sex of those Students whose age is greater than 24.

CREATE VIEW Student_view AS
SELECT sid, sname, level, age, sex
FROM Student
WHERE age>24;

Student

Sid	Sname	Level	Age	Sex
101	Harendra	Undergraduate	22	Male
102	Ramesh	Undergraduate	21	Male
103	Nirab	Graduate	25	Male
104	Pratibha	Undergraduate	20	Female
105	Samrita	Graduate	24	Female
106	Aastha	Undergraduate	22	Female
107	Rabindra	Graduate	28	Male
108	Abin	Graduate	26	Male
201	Bharat	Postgraduate	30	Male
202	Sohan	Postgraduate	32	Male

Now by executing this query we get following view (logical table);

Student_view

Sid	Sname	Level	Age	Sex
103	Nirab	Graduate	25	Male
107	Rabindra	Graduate	28	Male
108	Abin	Graduate	26	Male
201	Bharat	Postgraduate	30	Male
202	Sohan	Postgraduate	32	Male

Now any valid database operations can be performed in this view like in that of general table. It is a named specification of a result table. The specification is a SELECT statement that is executed whenever the view is referenced in an SQL statement. Consider a view to have columns and rows just like a base table. For retrieval, all views can be used just like base tables.

4 a) Define a first, second, and third normal forms with suitable examples.

Solution:

First Normal Form (1NF)

A relation is said to be in 1NF if and only if all domains of the relation contains only atomic (indivisible) values.

More simply a relation is in 1 NF if it does not have multi-valued attributes, composite attributes and their combinations. It states that the domain of an attribute must include only atomic (simple, indivisible) values.

Example: let's take an un-normalized relation containing composite attributes as,

Student

Sid	Sname	Subjects	
		Subject1	Subject2
1	Nitesh	DBMS	Graphics
2	Laxman	C	C++
3	Geeta	JAVA	.NET
4	Anisha	Simulation	SAD
5	Monika	Algebra	Calculus

The above table cannot be considered as an example of 1 NF, because it has repeating groups (two subject fields).Now convert this relation into 1 NF as,

Student

Sid	Sname	Subject1	Subject2
1	Nitesh	DBMS	Graphics

2	Laxman	C	C++
3	Geeta	JAVA	.NET
4	Anisha	Simulation	SAD
5	Monika	Algebra	Calculus

Fig: table in 1 NF

Example 2: let's take a relation that is not in 1 NF (containing multi-valued attributes)
Student

Sid	Sname	Address	Phone_No
1	Nitesh	Kalanki	9849145464 9813335467
2	Laxman	Balkhu	9841882345 099392844
3	Geeta	Kirtipur	9848334898
4	Anisha	Pokhara	9849283847
5	Monika	Ratopool	9840084732 9803267499

Now converting this relation into 1 NF by decomposing this relation into two relations as,

Student

Sid	Sname	Address
1	Nitesh	Kalanki
2	Laxman	Balkhu
3	Geeta	Kirtipur
4	Anisha	Pokhara
5	Monika	Ratopool

Phone

Sid	Phone_No
1	9849145464
1	9813335467
2	9841882345
2	099392844
3	9848334898
4	9849283847
5	9840084732
5	9803267499

Fig: Relations in 1 NF

Second Normal Form (2NF)

A relation is said to be in 2NF if and only if

- ☞ It is already in 1NF and
- ☞ Every non-prime attribute is fully dependent on the primary key of the relation.

Speaking inversely, if a table has some attributes which is partially dependant on the primary key of that table then it is not in 2NF.
Example: let's take a relation in 1 NF but not in 2 NF as

Employee-Department

Emp-Id	Emp-Name	Emp-Salary	Dept-No	Dept-Name
1	Bhupi	40000	D1	BBA
1	Bhupi	40000	D2	CSIT
2	Bindu	30000	D3	BBS
3	Arjun	60000	D1	CSIT

In the above relation {Emp-Id, Dept-No} is the primary key. Emp-Name, Emp-Salary and Dept-Name all depend upon {Emp-Id, Dept-No}. Again Emp-Id→Emp-Name, Emp-Id→Emp-Salary and Dept-No→Dept-Name, thus there also occur partial dependency. Due to which this relation is not in 2 NF.
Now converting this relation into 2 NF by decomposing this relation into three relations as,

Employee

Emp-Id	**Emp-Name**	**Emp-Salary**
1	Bhupi	40000
2	Bindu	30000
3	Arjun	60000

Emp-Dept

Emp-Id	***Dept-No***
1	*D1*
1	*D2*
2	*D3*
3	*D1*

Department

Dept-No	***Dept-Name***
D1	BBA
D2	CSIT
D3	BBS

Fig: Relations in 2 NF

Third Normal Form (3NF)

A relation is said to be in 3NF if and only if

- ☞ it is already in 2NF and
- ☞ Every non-prime attribute is **non-transitively** dependent on the primary key.

Speaking inversely, if a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.

Example:

Student

S-Id	S-Name	Age	Sex	Hostel-Name
1	Laxmi	21	Female	White house
2	Binita	22	Female	White house

3	Rajesh	32	Male	Red house
4	Aayan	21	Male	Red house

Here S-Id→S-Name, S-Id→Age, S-Id→Sex, S-Id→Hostel-Name and also Sex→Hostel-Name

This last dependency was not originally specified but we have derived it. The derived dependency is called a transitive dependency**.** In such a case the relation should be broken into two relations as,

Student

S-Id	S-Name	Age	Sex
1	Laxmi	21	Female
2	Binita	22	Female
3	Rajesh	32	Male
4	Aayan	21	Male

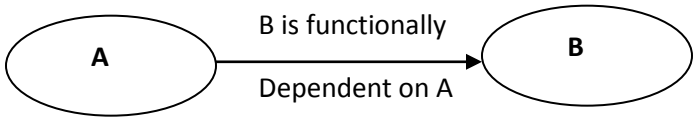
Hostel

Sex	Hostel-Name
Female	White house
Male	Red house

Fig: Relations in 3 NF

b) What is a functional dependency? When are two sets of functional dependencies equivalent? How can we determine their equivalence?

Solution: Functional Dependencies are fundamental to the process of normalization **Functional Dependency** describes the relationship between attributes (columns) in a table. For example, if **A** and **B** are attributes of a table, **B** is functionally dependent on **A**, if each value of **A** is associated with exactly one value of **B** (so, you can say, '**A** functionally determines **B**').



Determinant

Fig: Functional dependency between A and B

Two sets of functional dependencies *E* and *F* are equivalent if *E*+ = *F*+. Therefore, equivalence means that every FD in *E* can be inferred from *F*, and every FD in *F* can be inferred from *E*; that is, *E* is equivalent to *F* if both the conditions – *E* covers *F* **and** *F* covers *E* – hold.

For example:

Let's take two set of functional dependencies F={A→C, AC→D, E→AD, E→H} and G={A→CD, E→AH}

Soln:

Test for G is covered by F

{A}+={A, C, D} which covers A→CD in G

{E}+={E, A, D, H,} which cover E→AH in G

Hence G is covered by F

Also test for F is covered by G

{A}+={A, C, D} which covers A→C in F

{A, C}+={A, C, D}which cover AC→D in F

{E}+={E, A, D, H, C}which cover E→AD, E→H in F

Hence F is covered by G

This show that F is covered by G and G is covered by F hence these two ser of functional dependences are equivalent.

5. a) Discuss the ACID properties of a database transaction with suitable example.

Solution: The ACID properties are explained as follows.

Atomicity

Either all operations of the transaction are reflected properly in the database, or none are. The Atomicity property of a transaction implies that it will run to completion as an indivisible unit, at the end of which either no changes have occurred to the database or the database has been changed in a consistent manner. The basic idea behind ensuring atomicity is as follows. The database keeps a track of the old values of any database on which a transaction performs a write, and if the transaction does not complete its execution, the old values are restored to make it appear as though the transaction never executed. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity.

Consistency

The consistency property ensures that any transaction will bring the database from one valid state to another. Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database. The consistency property of a transaction implies that if the database was in a consistent state before the start of a transaction, then on termination of the transaction, the database will also be in a consistent state. Ensuring consistency for an individual transaction is the responsibility of the application Manager who codes the transaction. Note that during the execution of the transaction the state of the database becomes

inconsistent. Such an inconsistent state of the database should not be visible to the users or other concurrently running transactions.

Isolation

Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions *Ti* and *Tj*, it appears to *Ti* that either *Tj* finished execution before *Ti* started, or *Tj* started execution after *Ti* finished. Thus, each transaction is unaware of other transactions executing concurrently in the system. This means in case of concurrent execution of transaction, execution of one transaction should not interfare execution of another transaction. The isolation property of a transaction ensures that the concurrent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in same order. Thus, in a way it means that the actions performed by a transaction will be isolated or hidden from outside the transaction until the transaction terminates. This property gives the transaction a measure of relative independence. Ensuring the isolation property is the responsibility of a component of a database system called the Concurrency Control Component.

Durability

After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. 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. Durability can be guaranteed by ensuring that the updates carried out by the transaction have been written to the disk before the transaction completes and information about updates carried out by the transaction and written to the disk is sufficient to enable the database to re-construct the updates when the database system is restored after the failure. Ensuring durability is the responsibility of the component of the DBMS called the Recovery Management Component.

b) Describe the serial and serializable schedule? Why serializable schedule is consider correct?

Solution: A schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle then next transaction is executed. Transactions are ordered one after other. This type of schedule is called *serial schedule* as transactions are executed in a serial manner. In a serial

schedule only one transaction is active at a time. The commit (or abort) of one transaction initiates execution of the next transaction. Serial Schedule limit concurrency if one transaction waits for I/O operation to complete, the CPU cannot be switched to some other transaction. Hence serial schedules are considered unacceptable in practice.

T1	T2
Read(A) A= A-50 Write(A) Read(B) B=B+50 Write(B)	
	Read(A) Temp= A*0.1 A=A-Temp Write(A) Read(B) B= B + Temp Write(B)

Fig: Serial schedule

Serializable schedule:

A given non serial schedule of n transactions are serializable if it is equivalent to some serial schedule.That is if a non-serial schedule produce the same result as of the serial schedule then the given non-serial schedule is said to be serializable.A schedule that is not serializable is called a non-serializable. The main objective of serializability is to search non-serial schedules that allow transaction to execute concurrently without interfering one another transaction and produce the result that could be produced by a serial execution. In serializability, ordering of read/write is important. If two transactions only read data item they do not conflict and order is not important. If two transactions either read or write completely separate data items, they do not conflict and order is not important. If one transaction writes a data item and another reads or writes same data items, then order of execution is important.

6. a) How does the granularity of data items affects the performance of concurrency control? What factors affect selections of granularity size for data items?

Solution: In all concurrency control schemes, we have used each individual data item as the unit on which synchronization is performed. However, it would be advantageous to group several data items and to treat them as one individual unit. For example, if a transaction Ti needs to access the entire database and it uses a locking protocol, then Ti must lock each item in the database, which is time consuming process. Hence it would be better if Ti would issue a single lock request to lock the entire database. On the other hand if transaction Ti needs to access only a few data items, it should not be required to lock the entire database. A data item can be one of the following.

1. Database Field 2.Database record 3.Disk block 4.Whole file
   5. Whole database

The size of database item is often called the data item granularity. Fine granularity refers to overall item size where as coarse granularity refers to large item sizes. The best item size depends on the type transaction. Hierarchy of data granularities, where the small granularities are nested within larger ones, can be represented graphically a tree. In the tree, each node represents independent data item, non-leaf node of the multiple granularity tree represents the data associated with its descendents.

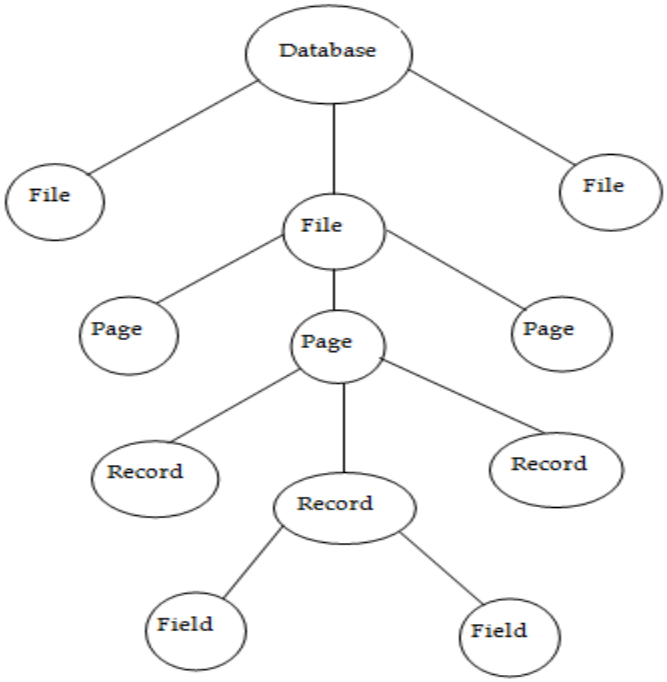


Figure: Hierarchy of Data Granurality

b) Describe the two-phase commit protocol for database transaction.

Solution: The two-phase locking protocol is used to ensure the serializability in Database. This protocol is implemented in two phase. *Growing Phase* and *Shrinking Phase***.**

- **Growing Phase**: In this phase we put read or write lock based on need on the data. In this phase we does not release any lock. Remember that all lock operation must preced first unloch operation appeared in a transaction.
- **Shrinking Phase:** This phase is just reverse of growing phase. In this phase we release read and write lock but doesn't put any lock on data. Unlock operations can only appear after last lock operation.

For a transaction these two phases must be mutually exclusive. This means, during locking phase unlocking phase must not start and during unlocking phase locking phase must not begin. It can be proved that, if every transaction in a schedule follows the 2PL, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedule any more. If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the growing phase, and downgrading of locks (from write-locked to read-locked) must be done in the shrinking phase. Hence, a read_lock(x) operation that downgrades an already held write lock on x can appear only in the shrinking phase and a write_lock(x) operation that upgrades an already held read lock on x can appear only in the growing phase.

Consider the following two schedules, both of the schedules are equivalent, only the difference is first schedule does not follow 2PL locking protocol whereas second schedule follows it. Both schedules contain two transactions, T1 and T2. Transaction T1 adds 100 to both data items and transaction T2 multiplies both data items by 2. Assume intial value of both data items (x and y) is 50. In schedule S1 final value of data items x and y is 300 and 250 respectively which is not coorect beacuase it is not equivalent to any serial schedule containing T1 and T2. But, in schedule S2 final value of both data items is 300. This is correct because it is erquivalent to serial schedule T1→T2. From this observation, we can conclude that Shedule S1 does not follows 2PL locking protocol therefore it is not serializable but schedule S2 follows 2PL protocol and hence it is serializable.

T1	T2	Data Items Values
Lock(x)		
Read(x)		x←50
x=x+100		x←150
Write(x)		T2 writes x←150
Unlock(x)		
	Lock(x)	
	x=x*2	x←300
	Wtite(x)	T2 writes x←300
	Unlock(x)	
	Lock(y)	
	Read(y)	y=50
	y=y*2	y=100
	Write(y)	T2 writes y←150
	Unlock(y)	
Lock(y)		
Read(y)		
y=y+100		y←150
Write(y)		y←250
Unlock(y)		T1 writes y←250

Schedule S1

T1	T2	Data Items Values
Lock(x)		
Read(x)		x←50
x=x+100		x←150
Write(x)		T1 writes x←150
Lock(y)		
Unlock(x)		
	Lock(x)	
	Read(x)	x←150
	x=x*2	x←300
	Wtite(x)	T2 writes x←300
Read(y)		y=50
y=y+100		y=150
Write(y)		T1 writes y←150
Unlock(y)		
	Lock(y)	
	Unlock(x)	
	Read(y)	y←150
	y=y*2	y←300
	Write(y)	T2 writes y←300
	Unlock(y)	

Schedule S2

TU **2067**

1. **Answer the following questions is short**.

a. Advantage of DBMS approach over file system approach.

Solution: Keeping organizational information in a file-processing system has a number of major disadvantages. Theses drawbacks of flat file systems are solved by database management systems. Drawbacks of file processing systems or we can say advantages of database management systems are described below:

Data redundancy: Data redundancy means duplication of same data or data files in different places. Flat file systems are suffered from the problem of high data redundancy. For example, record (such as student id, name, level, program, section etc) of a student may appear in library data files as well as examination data files. This redundancy leads to higher storage and access cost. On the other hand database management systems can greatly reduce the problem of data redundancy. Note that DBMS cannot remove data redundancy problem completely.

Data inconsistency: Data inconsistency is side effect of data redundancy. Data is said to be inconsistent if various copies of the same data may no longer agree. Data inconstancy occurs if changed data is reflected in data files in one place but not elsewhere in the system.

Data isolation: Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult in flat file systems.

Difficulty in accessing data: File processing systems do not allow required data to be retrieved in efficient and convenient way. For example, assume we already have program to generate the list of books on the basis of subject. Now, if we need to generate the list of books on the basis of author name, either we need to extract the data from book data files manually or we should request the programmer to write a program to retrieve required data from the book data file. Both of the alternatives are not satisfactory. First alternative is time consuming and the second alternative is tedious and costly because requesting a programmer every time to write a new program as we don't have application program to generate the required list of data is not good idea. But in database systems it is very easy to write general programs to generate different list on the basis of different criteria.

Integrity problems: Integrity means correctness of data before and after execution of a transaction. Integrity constraints are condition applied to the data. For example, if maximum salary in an organization is 150,000 then we have the integrity constraint "salary ≤ 150,000". Integrity constraints are important to maintain correctness of data. It plays vital to prevent users from doing mistakes. For example, if user mistakenly types 200,000 in place of 20,000 while transferring salary of an employee in his/her account, specified integrity constrain is violated and hence the system tell the user about the mistake.

Atomicity problems: Execution of transactions must be atomic. This means transactions must execute at its entirety or not at all. If execution of transaction is not atomic, it leaves database in incorrect sate.

Concurrent-access anomalies: Concurrent updates to same data by different transactions at the same time may result in inconsistent data.

Security problems: In database system we may create different user accounts and provide different authorization to different users. Thus we are able to hide certain information from some users

b. Differentiate between two-tier and three-tier client/server architecture.

Solution:

All projects are broadly divided into two types of applications **2 tier and 3 tier architecture**. Basically high level we can say that *2-tier architecture* is Client server application and *3-tier architecture* is Web based application. Below I am concentrating on the difference between Two-Tier and Three-Tier Architecture, what all advantages, disadvantages and practical examples.

Two-Tier Architecture

The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster. In this approach, the user interface and application programs are placed on the client side and the database system on the server side. This architecture is called two-tier architecture. The application programs that reside at the client side invoke the DBMS at the server side. The application program interface standards like Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) are used for interaction between client and server.

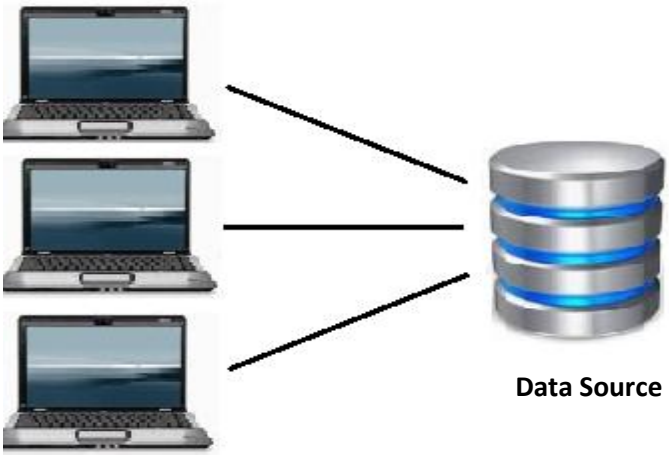


Fig: Two-Tier Architecture

The above figure shows the architecture of two-tier. Here the direct communication between client and server, there is no intermediate between client and server.

The Two-tier architecture is divided into two parts:

1) Client Application (Client Tier)

2) Database (Data Tier)

On client application side the code is written for saving the data in the SQL server database. Client sends the request to server and it process the request & send back with data. The main problem of two tier architecture is the server cannot respond multiple request same time, as a result it cause a data integrity issue.

Advantages:

1. Easy to maintain and modification is bit easy
2. Communication is faster

Disadvantages:

1. In two tier architecture application performance will be degrade upon increasing the users.
2. Cost-ineffective

Three-Tier Architecture

The three-tier architecture is primarily used for web-based applications. It adds intermediate layer known as application server (or web server) between the client and the database server. The client communicates with the application server, which in turn communicates with the database server. The application server stores the business rules (procedures and constraints) used for accessing data from database server. It checks the client's credentials before forwarding a request to database server. Hence, it improves database security. When a client requests for information, the application server accepts the request, processes it, and sends corresponding database commands to database server. The database server sends the result back to application server which is converted into GUI format and presented to the client. Three layers in the three tier architecture are as follows:

1) Client layer
2) Business layer
3) Data layer

1) Client layer: It is also called as *Presentation layer* which contains UI part of our application. This layer is used for the design purpose where data is presented to the

user or input is taken from the user. For example designing registration form which contains text box, label, button etc.

2) Business layer:

In this layer all business logic written likes validation of data, calculations, data insertion etc. This acts as a interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

3) Data layer:

In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.

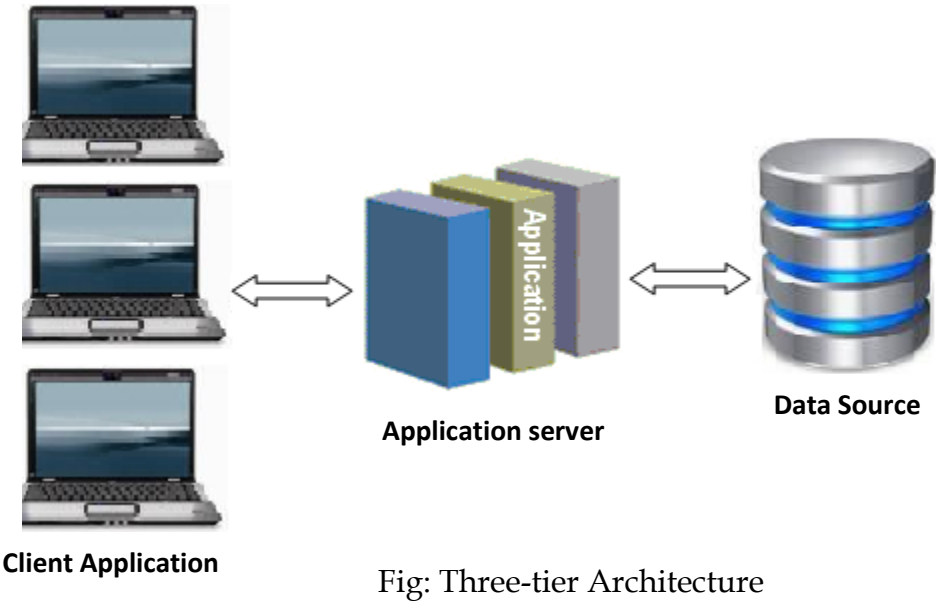


Fig: Three-tier Architecture

Advantages

1. High performance, lightweight persistent objects
2. Scalability – Each tier can scale horizontally
3. Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
4. High degree of flexibility in deployment platform and configuration
5. Better Re-use
6. Improve Data Integrity
7. Improved Security – Client is not direct access to database.
8. Easy to maintain and modification is bit easy, won't affect other modules
9. In three tier architecture application performance is good.

Disadvantages

1. Increase Complexity/Effort

c. What are weak entity, owner entity type and identifying relationship?

Solution: An entity set that does not possess sufficient attributes to form a primary key is called a **weak entity set**. One that does have a primary key is called a strong entity set. For example, the entity set transaction has attributes transaction number (Tno), date and amount. Different transactions on different accounts could share the same number. Therefore these are not sufficient to form a primary key (uniquely identify a transaction). Thus transaction is a weak entity set. Strong entity set on which existence of weak entity set depends is called **owner or identifying entity set.** The one-to-many relationship between weak entity set and its owner entity set is **called identifying relationship**.

In ER diagram weak entity sets are represented by double rectangle, identifying relationship is represented by double diamond and discriminator is represented by underlining the attribute(s) by dotted line as shown in the diagram below. It should be noted that participation of weak entity set is always total.

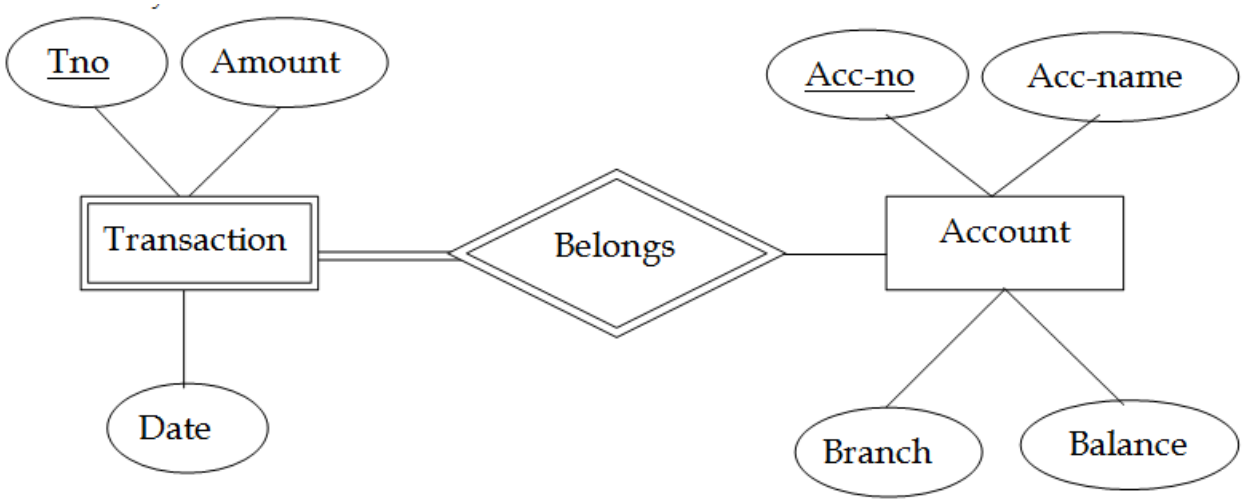


Illustration:

- Transaction is a weak entity. It is existence-dependent on account.
- The primary key of account is account-number.
- Transaction number (Tno) distinguishes transaction entities within the same account and is thus the discriminator.
- So the primary key for transaction would be {Tno, Acc-no}

d. The null value attribute and its uses.

Solution: SQL allows the use of NULL values to indicate absence of information about the value of an attribute. It has a special meaning in the database- the value of the column is not currently known but its value may be known at a later time.

A special comparison operator IS NULL is used to test a column value for NULL. It has following general format:

Value IS [NOT] NULL;

This comparison operator return *true* if value contains NULL, otherwise return *false.* The optional NOT reverses the result.

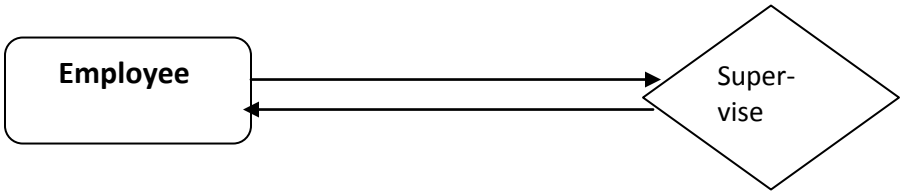
NULL is the value used to represent an unknown piece of data. Let's take a look at a simple example: a table containing the inventory for a fruit stand. Suppose that our inventory contains 10 apples, 3 oranges. We also stock plums, but our inventory information is incomplete and we don't know how many (if any) plums are in stock. Using the NULL value, we would have the inventory table as shown in below:

Fruit Stand Inventory

Item	**Quantity**
Apples	10
Oranges	3
Plums	NULL

e. Recursive relationship type with suitable example.

Solution: A recursive relationship can be defined as a relationship that is expressed about multiple records within one table. As an example if we take an employee table then there are some employees who are supervisor and some who are being supervised. This is the relationship of Supervisor and supervisee is called a recursive relationship.



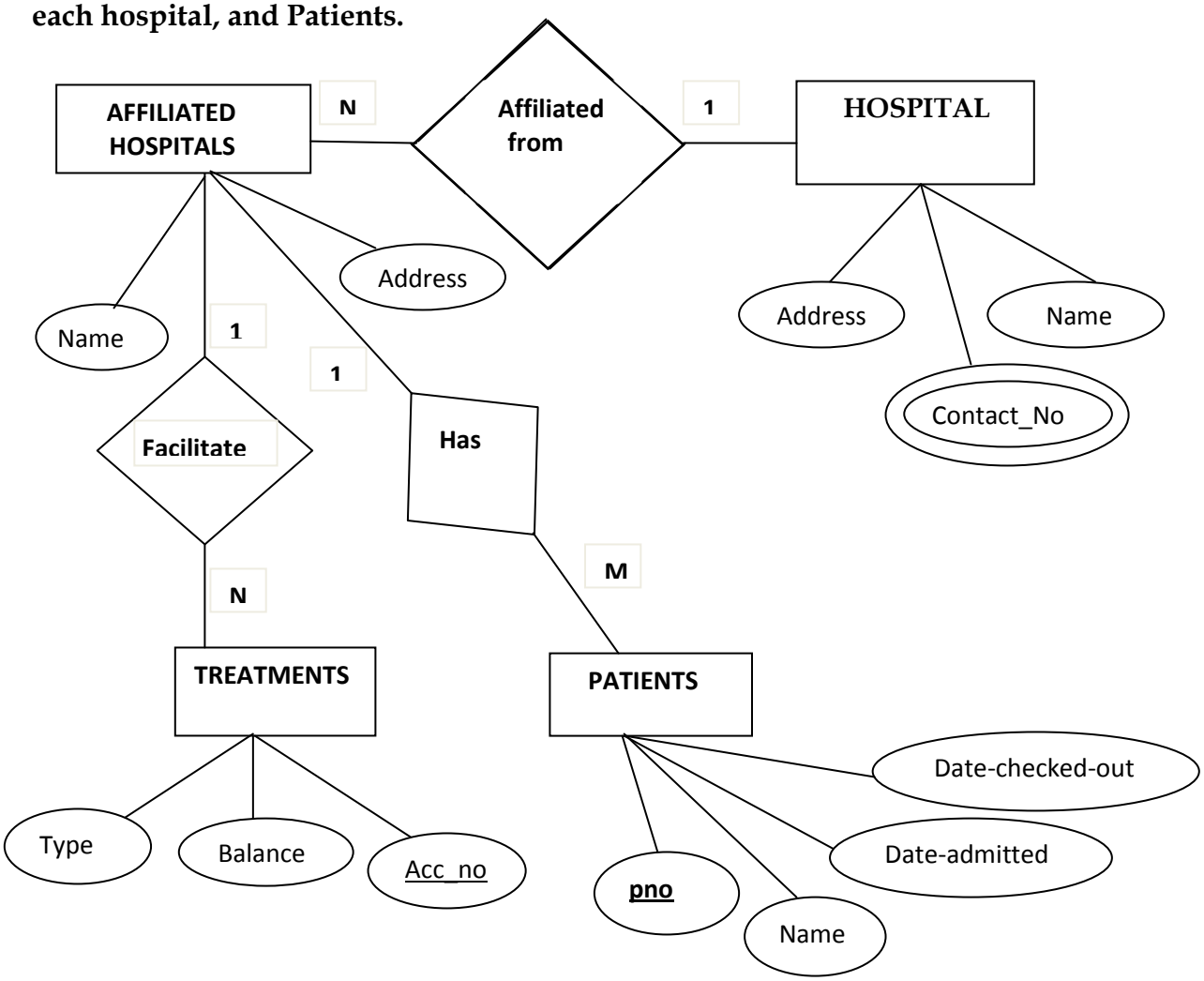
Consider other situations such as:

- a team plays a game against another team
- a person is a child of a person
- an organizational unit (e.g. department, division, branch, ...) comprises other units

- a course is a prerequisite for another course

2.

a. Draw an ER diagram for a database showing Hospital system. The Hospital maintains data about Affiliated Hospitals, type of Treatments facilities given at each hospital, and Patients.



b. What is join operation? Differentiate between equijoin and natural join with suitable example.

Solution: What is join operation?

Join is a special form of cross product of two tables. It is a binary operation that allows combining certain selections and a Cartesian product into one operation. The join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes. Following are the different types of joins:

1. Theta Join 2. Equi Join 3. Semi Join 4. Natural Join 5. Outer Joins

Natural join: Natural join (⋈) is a binary operator that is written as ($R \bowtie S$) where R and S are relations. The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names.
It is same as equijoin but the difference is that in natural join, the common attribute appears only once and at least one attribute must be common.

For example consider the tables *Employee* and *Dept* and their natural join:

Employee

e-id	e-name	Dept
11	Bhupi	Computer
13	Anju	Finance
43	Manju	Computer
54	Nisha	Finance

Department

Dept	Manager
Computer	Anisha
Finance	Manisha
Production	Umesh

Employee ⋈epartment

e-id	e-name	Dept	manager
11	Bhupi	Computer	Anisha
13	Anju	Finance	Manisha
43	Manju	Computer	Anisha
54	Nisha	Finance	Manisha

Equi-join: It is special case of conditional join where the conditions consist only of equalities. Unlike natural join, here relations may have different attributes and if a common attribute occur then such attribute occur two times in the resulting equi-join operation.

Example:

Employee $\bowtie_{\text{Employee.Dept=Department.Dept}}$ Department

e-id	e-name	E.Dept	manager	D.Dept
11	Bhupi	Computer	Anisha	Computer
13	Anju	Finance	Manisha	Finance
43	Manju	Computer	Anisha	Computer
54	Nisha	Finance	Manisha	Finance

3. **Assume database about Company**

EMPLOYEE (ss#, name)
COMPANY (cname, address)
WORKS (ss#, cname)
SUPERVISES (supervisor_ss#, employee_ss#)

Write relational algebra and SQL queries for each of the following cases.

a. Find the names of supervisors that work in companies whose address equals '**Kathmandu'**

```
SELECT name
FROM Employee e, Supervise s, Works w, Company c
WHERE C.address='Kathmandu' AND e.ss#=s.supervisor_ss# AND
s.supervisor_ss# = w.ss# AND w.cname=c.cname;
```

b. Find the names of all the companies who have more than 4 supervisors.

```
SELECT cname, COUNT (supervisor_ss#) AS nos
FROM Company c, Works w, Superviser s
WHERE C.cname=w.cname AND w.ss#=s.Supervisor_ss#
GROUP BY cname
HAVING nos>4;
```

c. Find the name of the supervisor who has the largest number of employees.

```
SELECT name, MAX (noe)
FROM (SELECT name, supervisor_ss#, COUNT (employee_ss#) AS noe
        FROM Employee e, Supervise s
            WHERE e.ss#=s.supervisor_ss#
                GROUP BY supervisor_ss#);
```

d. How can define view in SQL? Explain the problems that may arise when one attempts to update a view.

Solution: A database view is a logical table. It does not physically store data like tables but represent data stored in underlying tables in different formats. A view does not require desk space and we can use view in most places where a table can be used. Since the views are derived from other tables thus when the data in its source tables are updated, the view reflects the updates as well. They also can be used by DBA to enforce database security.

Advantages of Views:

- ☞ Database security: view allows users to access only those sections of database that directly concerns them.
- ☞ View provides data independence.
- ☞ Easier querying
- ☞ Shielding from change
- ☞ Views provide group of users to access the data according to their criteria.
- ☞ Vies allow the same data to be seen by different users in different ways at the same time.

When the column of a view is directly derived from the column of a base table, that column inherits any constraints that apply to the column of the base table. For example, if a view includes a foreign key of its base table, INSERT and UPDATE operations using that view are subject to the same referential constraints as the base table.

Syntax for creating view is:

CREATE VIEW <view name> AS <query expression>

Where, <query expression> is any legal query expression.

Problems that may arise when one attempts to update a view:

Although views are a useful tool for queries, they present serious problems if we express updates, insertions, or deletions with them. The difficulty is that a modification to the database expressed in terms of a view must be translated to a modification to the actual relations in the logical model of the database. To illustrate the problem, consider a clerk who needs to see all loan data in the *loan* relation, except *loan-amount*. Let *loan-branch* be the view given to the clerk. We define this view as

Create view *loan-branch* **as**

$\Pi_{loan\text{-}number,\ branch\text{-}name}$ (*loan*)

Since we allow a view name to appear wherever a relation name is allowed, the clerk can write:

loan-branch $\leftarrow$ *loan-branch* $\cup$ {(L-37, "Perryridge")}

This insertion must be represented by an insertion into the relation *loan*, since *loan* is the actual relation from which the database system constructs the view *loan-branch*. However, to insert a tuple into *loan*, we must have some value for *amount*. There are two reasonable approaches to dealing with this insertion:

- Reject the insertion, and return an error message to the user.
- Insert a tuple (L-37, "Perryridge", *null*) into the *loan* relation.

Another problem with modification of the database through views occurs with a view such as

create view *loan-info* **as**

$\Pi_{customer\text{-}name,\ amount}(borrower \bowtie loan)$

This view lists the loan amount for each loan that any customer of the bank has. Consider the following insertion through this view:

loan-info ← *loan-info* ∪ {("Johnson", 1900)}

The only possible method of inserting tuples into the *borrower* and *loan* relations is to insert ("Johnson", *null*) into *borrower* and (*null*, *null*, 1900) into *loan*. Then, we obtain the relations shown in Figure 3.36. However, this update does not have the desired effect, since the view relation *loan-info* still does *not* include the tuple ("Johnson", 1900). Thus, there is no way to update the relations *borrower* and *loan* by using nulls to get the desired update on *loan-info*.

Because of problems such as these, modifications are generally not permitted on view relations, except in limited cases. Different database systems specify different conditions under which they permit updates on view relations;

4. What are different update anomalies? Explain each in with suitable examples.

Solution:

The terms "Update Anomalies" are called the problems which are the results from the un-normalized database in the Relational Database Management System (RDBMS). This is the common name given to anomalies. But if we are talking about the Update Anomalies it means we are talking about the Insertion Anomalies, Deletion Anomalies and Modification Anomalies. If these three anomalies are it means there is some inconsistency in our database. This will definitely create the problems while inserting, deleting and modifying the records in the data base entities called "tables". These three Update Anomalies are having different impact on our database. These are classified as mentioned below:-

- Insertion Anomalies create the problems when we are creating the inconsistency in the RDBMS database while inserting the records into the columns of the given table.

- Deletion Anomalies create the problems when we are deleting the records without taking care of the other portion of the database. It will create the confusion due to inconsistency in the database.
- Modification Anomalies are occurs when we are not able to modifying the records in the data base without taking care of the other facts.

Example

Consider the following relation

Eid	**Ename**	**Salary**	**Dno**	**Dname**
E01	Ram	23000	D1	IT
E02	Harry	28000	D2	Admin
E03	Ramila	22000	D1	IT
E04	Bharat	32000	D3	Finance
E04	Bharat	32000	D2	Admin

In the above relation if we want to insert information about newly hired employee to whom department is not assigned yet, we can insert the information because value of Dno cannot be null. This problem is called insertion anomaly.

If we want to delete information about department D2 without deleting information about employees in the department, we are not able to do this. Deleting department D2 causes deletion of employee E02 and E04 also. This problem is called deletion anomaly.

Changing the name of department number D1 from "IT" to "ICT" may cause this update to be made for all employees working on department D1. This problem is called update anomaly.

b. Define functional dependency. Describe the closure of a set of functional dependencies with an example.

Solution: Functional dependency plays a key role in differentiating good database design from bad database design. It describes the relationship between attributes (columns) in a table.

Let X and Y are attributes of a relation R. If each value of Y is associated with exactly one value of X then Y is said to be functionally dependent on X. it is denoted by X→Y.

So the functional dependency X → Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y. For any two tuples t1 and t2 in any relation instance r(R):

If t1[X] = t2[X], *then* t1[Y] = t2[Y]

Example: in the following relation

Customer (Cid, Cname, Age, salary)

- ☞ Cid → Cname

It is true because Cid uniquely determine the customer name even in the case of duplicate Cname.

- ☞ Cname→ Cid

It is false because the name of the customer may be same for different Cid. So Cname not uniquely determine the customer.

- ☞ Cid → Age [True]
- ☞ Cid → salary [True]
- ☞ Age→cid [False]

It is not sufficient to consider the given set of functional dependencies. Rather, we need to consider *all* functional dependencies that hold. We shall see that, given a set *F* of functional dependencies, we can prove that certain other functional dependencies hold. We say that such functional dependencies are "logically implied" by *F*. More formally, given a relational schema *R*, a functional dependency *f* on *R* is **logically implied** by a set of functional dependencies *F* on *R* if every relation instance *r*(*R*) that satisfies *F* also satisfies *f*.

In brief the set of dependencies that is logically implies by F is called the closure of F and it's closure is written as F^+.

We can find all of F+ by applying Armstrong's Axioms:

- ✓ if β ⊆ α then α → β or α →α (reflexive)
- ✓ if α → β then γ α →γ β (augmentation)
- ✓ if α →β and β →γ then α →γ (transitivity)

We can further simplify the the computation of F+ by using the following addition rule.

- ✓ if α → β holds and α → γ holds, then α → β γ (Additivity or union rule)
- ✓ if α → β γ holds then α → β holds and α → γ holds (projectivity/decomposion)
- ✓ if α → β holds and γ β →δ holds then α γ →δ holds (pseudotransitivity)

Example: Let R = (A, B, C, G, H, I)

F= {A→B, A→C, CG→H, CG→I, B→H}

Compute closure of F (F^+).

Sol^n:- Since A→B, B→H then A→H [by transitivity rule]

Since A→B, A→C then A→BC [by union rule]

Since A→C then AG→CG [by augmentation rule]

Since AG→CG, CG→I then AG→I [by transitivity rule]

Since CG →H, CG→I then CG→HI [by union rule]

Hence, F^+ = {A→A, B→B, C→C, H→H, G→G, I→I, A→B, A→C, CG→H, G→I, CG→HI, B→H, A→H, AG→I, CG→HI}

5. a. Draw a state diagram, and discuss the typical state that a transaction goes through during transaction.

Solution: Whenever a transaction is submitted to a DBMS for execution, either it executes successfully or fails due to some reasons. During its execution, a transaction passes through various states that are *active*, *partially committed*, *committed*, *failed*, and *aborted*. Figure 10.1 shows the state transition diagram that describes how a transaction passes through various states during its execution.

- **Active State:** In this state the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed State:** When a transaction executes its final operation, it is said to be in this state. After execution of all operations, the database system performs some checks e.g. the consistency state of database after applying output of transaction onto the database.
- **Failed State:** If any check made by database recovery system fails, the transaction is said to be in failed state, from where it can no longer proceed further.
- **Aborted:** If any of checks fails and transaction reached in failed state, the recovery manager rolls back all its write operation on the database to make database in the state where it was prior to start of execution of transaction. Transactions in this state are called aborted. Database recovery module can select one of the two operations after a transaction aborts:
  - Re-start the transaction
  - Kill the transaction
- **Committed State:** If transaction executes all its operations successfully it is said to be committed. All its effects are now permanently made on database system.

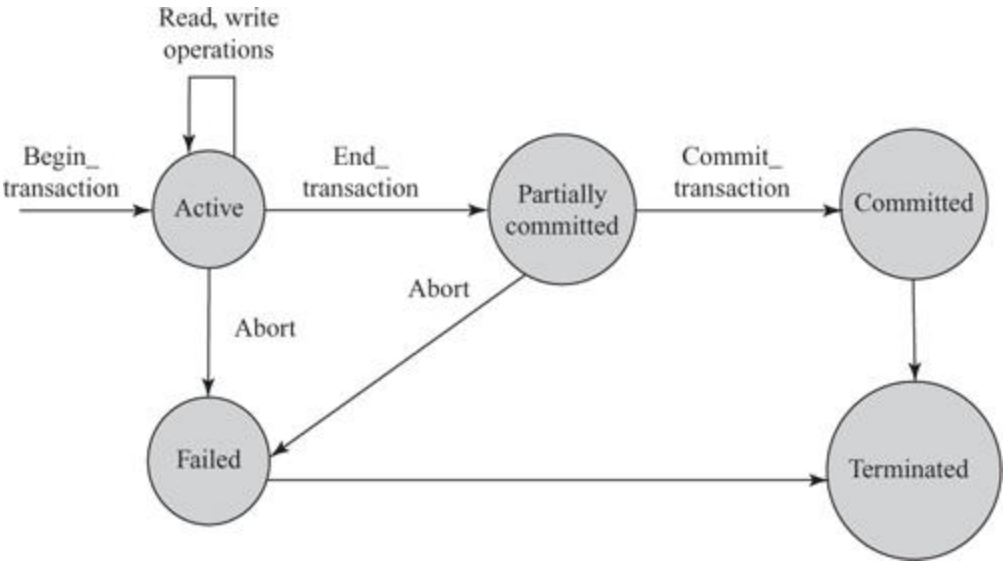
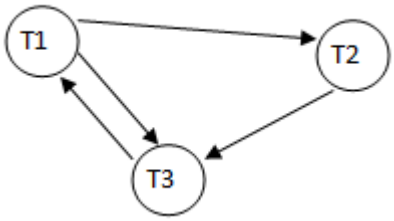


Fig: State diagram of transaction

b. Which of the following schedule is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules.

a. r1(x); r3(x); w1(x); r2(x); w3(x);

T1	T2	T3
r(x)		
		r(x)
w(x)		
	r(x)	
		w(x)



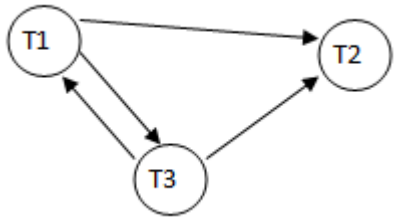
Dependence graph

Since the above precedence graph is cyclic hence given schedule is not conflict serializable. And its serial schedule cannot be determined.

ii). r1(x); r3(x); w3(x); w1(x); r2(x);

T1	T2	T3

r(x)		
		r(x)
		w(x)
w(x)		
	r(x)	

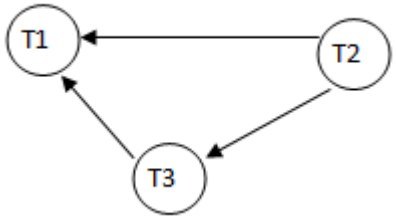


Dependence graph

Since the above precedence graph is cyclic hence given schedule is not conflict serializable. And its serial schedule cannot be determined.

3) r3(x);r2(x); r1(x);w3(x);w1(x);

T1	T2	T3
		r(x)
	r(x)	
		w(x)
r(x)		
w(x)		



Dependence graph

Since the above precedence graph is not cyclic hence given schedule is conflict serializable. And its serial schedule is T2→T3→T1

6. a) Discuss the problems of deadlock and starvation, and the different approaches to dealing with these problems.

Solution: A system is in a deadlock state if there exists a set of transactions such that every transaction in the set in waiting for data item that is locked by another transaction in the set. There exists a set of waiting transactions {T0, T1,…….Tn} such that T0 is waiting for data item that is held by T1, T1 is waiting for a data item that is held by T2, Tn-1 is waiting for a data item that is held by Tn, and Tn is waiting for a data item that is held by T0. None of the transactions can make progress in such a situation.

The different approaches to dealing with these problems are:

☞ **Wait-Die Scheme:** It is a non-preemptive technique for deadlock prevention. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j (That is T_i is older than T_j), otherwise T_i is rolled back (dies). Thus this scheme prefers younger transactions.

If TS(Ti)< TS(Tj) //Ti older tha Tj

then

Ti waits

else

Ti dies

For example, suppose that transaction T2, T3 and T4 have timestamps 5, 10 and 15 respectively. If T2 requests a data item held by T3, then T2 will wait. If T4 requests a data item held by T3, then T4 will be rolled back.

Wound-Wait Scheme: It is a preemptive technique for deadlock prevention. It is a counterpart to the wait-die scheme. When transaction T_i requests a data item currently held by transactionT_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j (i.e Ti younger than Tj), otherwise T_j is rolled back (T_j is wounded by T_i). This scheme prefers older transactions.

if TS(Ti)< TS(Tj)

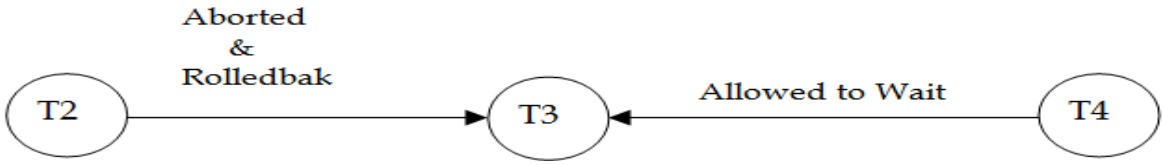
then

Tj is wounded (rolled back)

else

Ti waits

In above example, if transaction T2 requests a data item held by transaction T3, then the data item will be preempted from T3 and T3 will be aborted and rolledback. If T4 requests a data item held by T3, then T4 will wait.



Another group of protocol that prevent deadlock do not require timestamps are *No Waiting* (NW) and *Cautious Waiting* (CW) algorithms.

- **No Waiting Algorithm**: If a transaction is unable to obtain a lock on a data item, it is immediately aborted and then restarted after a certain time delay without checking whether a deadlock will actually occur or not. This can cause transactions to abort and restart needlessly.

- **Cautious Waiting Algorithm:** This is proposed to avoid needless restart in case of no waiting algorithm, if transaction T1 tries to lock a data item x, but is not able to do so because x is locked by some other transaction T2 a exclusive lock. If T2 is not blocked on some other locked data item, then T1 is blocked and allowed to wait; otherwise abort T1.

b) Describe write-ahead logging protocol

Solution: Write-ahead log (WAL) protocol guarantees that no data modifications are written to disk before the associated log record is written to disk. This maintains the ACID properties for a transaction. At the time a modification is made to a page in the buffer, a log record is built in the log cache that records the modification. This log record must be written to disk before the associated dirty page is flushed from the buffer cache to disk. If the dirty page is flushed before the log record is written, the dirty page creates a modification on the disk that cannot be rolled back if the transaction fails before the log record is written to disk. WAL states that

- For Undo: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).

- For Redo: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

TU 2069

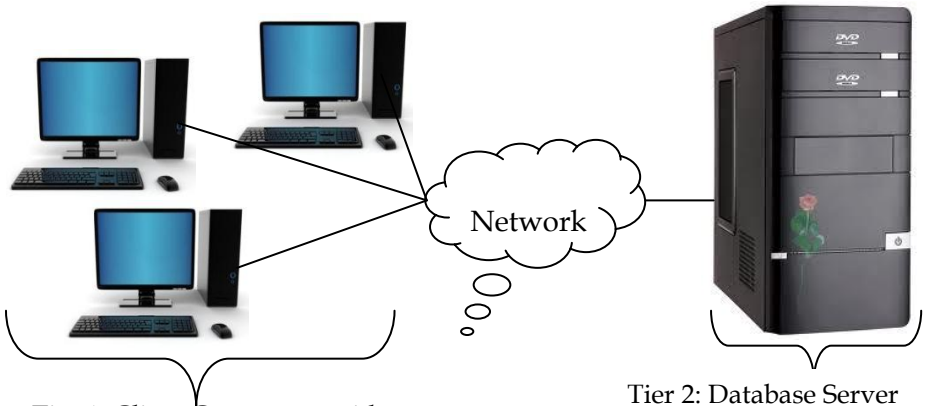
1. Answer the following questions in short:

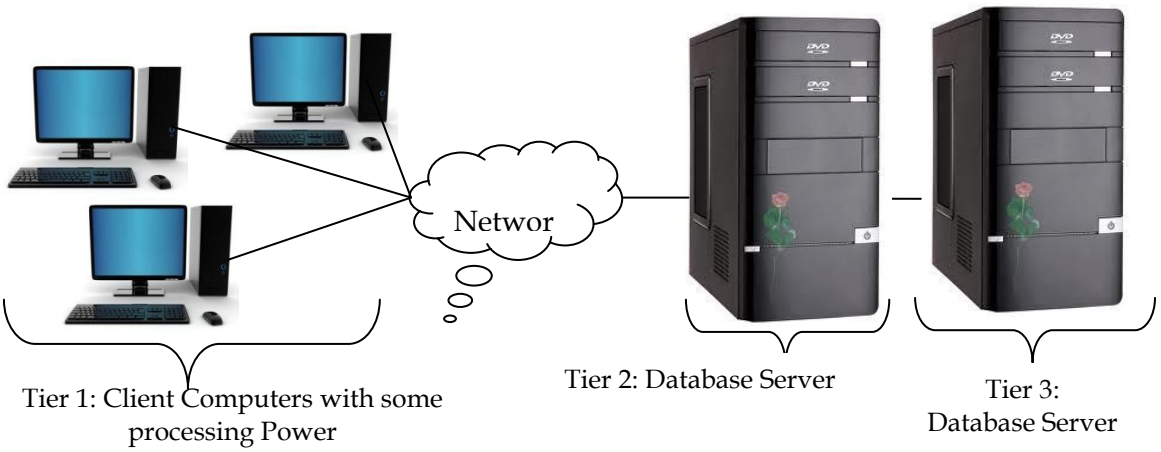
a. Differentiate between two-tier and three tier client/server architecture.

Solution: In client/server architecture, the processing power of the computer system at the user's end is utilized by processing the user-interface on that system. A client is a computer system that sends request to the server connected to the network, and a server is a computer system that receives the request, processes it, and returns the requested information back to the client. Client and server are usually present at different sites. The end users (remote database users) work on client computer system and database system runs on the server. The client machines have user interfaces that help users to utilize the servers. It also provides users the local processing power to run local applications on the client side. There are two approaches to implement client/server architecture: two-tier architecture and three-tier architecture.

In the first approach, the user interface and application programs are placed on the client side and the database system on the server side. This architecture is called two-tier architecture. The application programs that reside at the client side invoke the DBMS at the server side. The application program interface standards like Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) are used for interaction between client and server.

The second approach, that is, three-tier architecture is primarily used for web-based applications. It adds intermediate layer known as application server (or web server) between the client and the database server. The client communicates with the application server, which in turn communicates with the database server. The application server stores the business rules (procedures and constraints) used for accessing data from database server. It checks the client's credentials before forwarding a request to database server. Hence, it improves database security. When a client requests for information, the application server accepts the request, processes it, and sends corresponding database commands to database server. The database server sends the result back to application server which is converted into GUI format and presented to the client.





b. The null value attribute and its uses.

Solution: SQL allows the use of NULL values to indicate absence of information about the value of an attribute. It has a special meaning in the database- the value of the column is not currently known but its value may be known at a later time.

A special comparison operator IS NULL is used to test a column value for NULL. It has following general format:

Value IS [NOT] NULL;

This comparison operator return *true* if value contains NULL, otherwise return *false.* The optional NOT reverses the result.

NULL is the value used to represent an unknown piece of data. Let's take a look at a simple example: a table containing the inventory for a fruit stand. Suppose that our inventory contains 10 apples, 3 oranges. We also stock plums, but our inventory information is incomplete and we don't know how many (if any) plums are in stock. Using the NULL value, we would have the inventory table as shown in below:

Fruit Stand Inventory

Item	**Quantity**
Apples	10
Oranges	3
Plums	NULL

c. Differentiate between logical data independence and physical data independence.

Solution:

Logical Data Independence

The capacity to change the conceptual (logical level) schema without having to change associated application programs is called logical data independence. If the underlying conceptual schema is changed, the definition of a view relation can be modified so that the same relation is computed as before. Database administrator is responsible for redefining view level schema.

When modification is done to the conceptual schema (i.e tables) only the external/conceptual mapping need to be changed, if the DBMS fully supports the concept of data independence.

Physical Data Independence

The capacity to change the internal schema without affecting application programs is called physical data independence. This means we can change physical level storage details such as file structure; indexes as long as conceptual schema remains same without altering associated application programs. But performance may be affected due to changes in physical level. It is the responsibility of the DBA to manage such changes.

d. When is the concept of weak entity used in data modeling?

Solution: An entity set that does not possess sufficient attributes to form a primary key is called a **weak entity set**. One that does have a primary key is called a **strong entity set**. For example, the entity set transaction has attributes transaction number (Tno), date and amount. Different transactions on different accounts could share the same number. Therefore these are not sufficient to form a primary key (uniquely identify a transaction). Thus transaction is a weak entity set. Strong entity set on which existence of weak entity set depends is called **owner or identifying entity set**. The one-to-many relationship between weak entity set and its owner entity set is called **identifying relationship**. A weak entity set does not have a primary key, but we need a means of distinguishing among the entities. The primary key of owner entity set and discriminator of the weak entity set allows this distinction to be made. An attribute of weak entity set that is use in combination with primary key of the strong entity set to identify the weak entity set uniquely is called **discriminator** (partial key).

In ER diagram weak entity sets are represented by double rectangle, identifying relationship is represented by double diamond and discriminator is represented by underlining the attribute(s) by dotted line as shown in the diagram below. It should be noted that participation of weak entity set is always total.

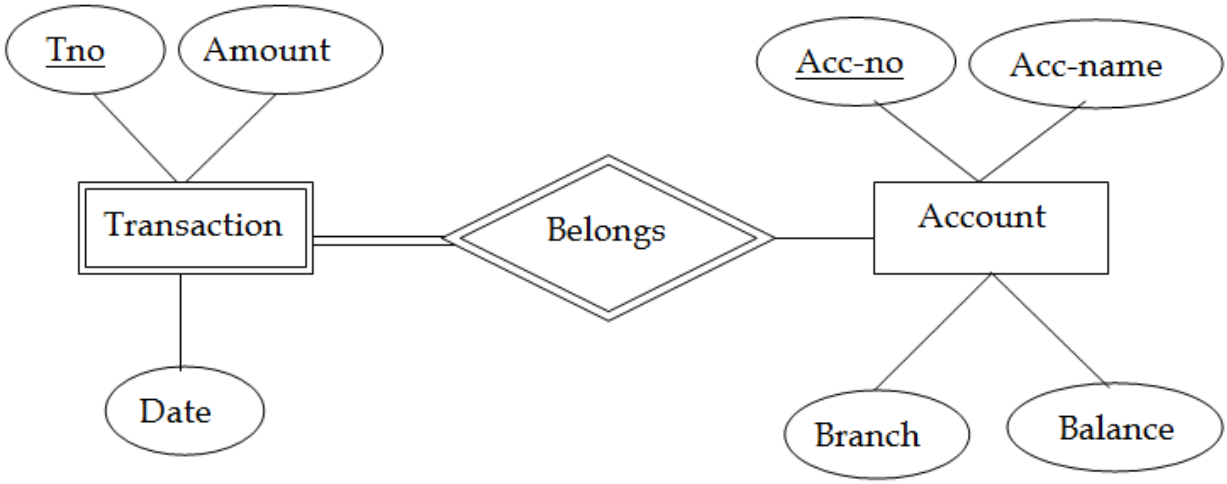
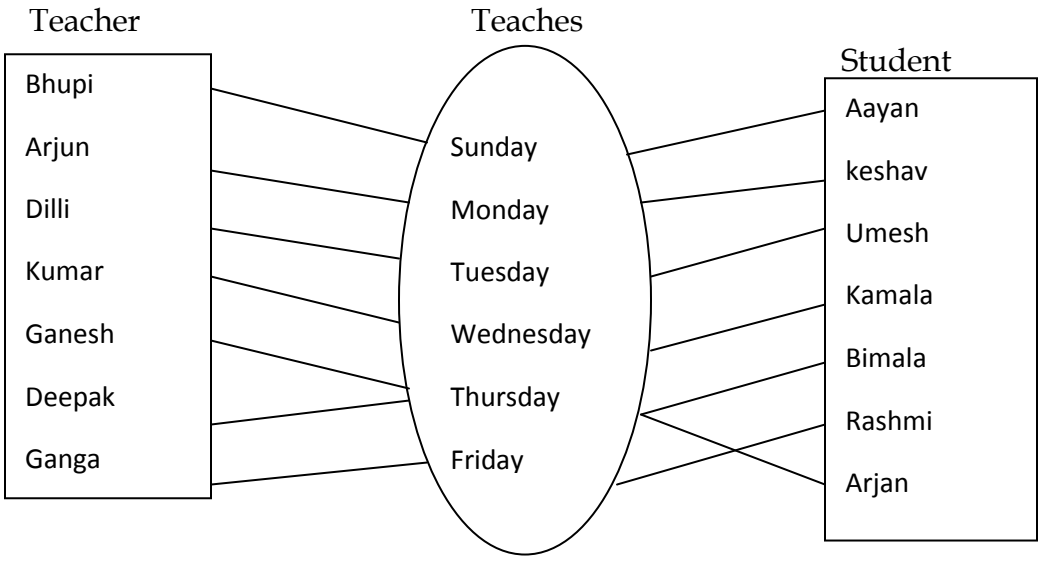


Illustration:

- Transaction is a weak entity. It is existence-dependent on account.
- The primary key of account is account-number.
- Transaction number (Tno) distinguishes transaction entities within the same account and is thus the discriminator.
- So the primary key for transaction would be {Tno, Acc-no}

e. The difference among a relationship instance, a relationship type, and a relationship set.

Solution: The relationship among Particular two or more entities of a entity set is called relationship instance.
The relationship among two or more entity sets is called relationship set.
A relationship type is an abstraction of a relationship i.e. a set of relationships instances sharing common attributes.

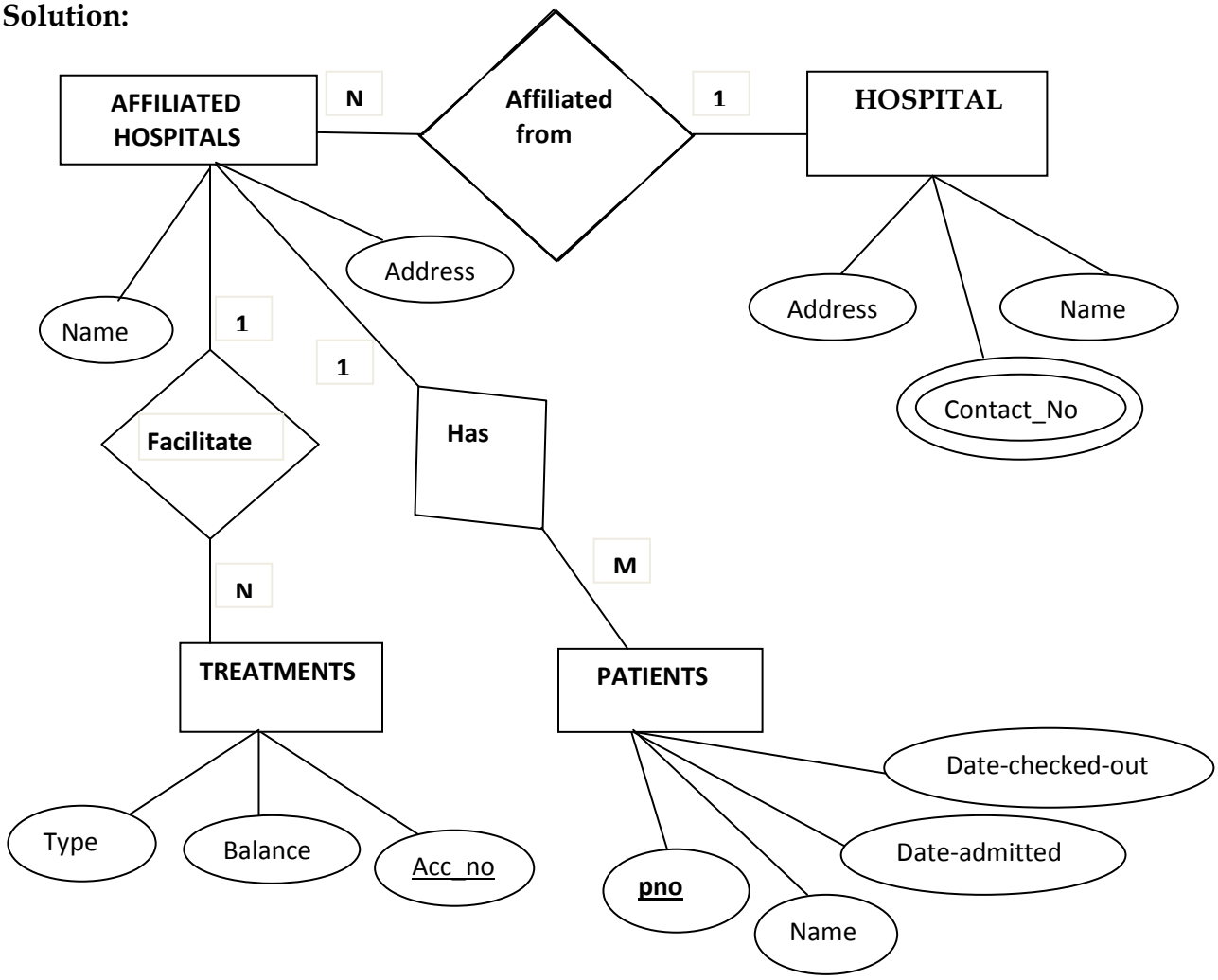


In the above fig teacher teaches student is called relationship set. If we take a particular instance i.e. Bhupi teaches at Sunday to Aayan is called relationship instance. And finally if each teacher uses same attributes and also each student uses same attributes then association among them is called relationship type. The relationship type may be one of the below:

one to one
one to many
many to one
many to many

2. (a) Draw an ER diagram for a database showing Hospital system. The Hospital maintains data about Affiliated Hospitals, type of Treatments, facilities given at each hospital and Patients.

Solution:



(b) In what sense does relational calculus differ from relational algebra, and in what sense they are similar?

Solution: The difference between relational calculus and relational algebra are listed below:

1. Relational algebra operations manipulate some relations and provide some expression in the form of queries where as relational calculus are formed queries on the basis of pairs of expressions.
2. RA have operator like join, union, intersection, division, difference, projection, selection etc. whereas RC has tuples and domain oriented expressions.
3. RA is procedural language where as RC is non procedural query system.
4. There is modification which is easy in queries in RA than the RC.
5. Relational algebra is easy to manipulate and understand than RC.
6. RA queries are more powerful than the RC.

The similarity between relational calculus and relational algebra are listed below:

1. Both relational algebra and relational calculus are formal languages associated with relational model that are used to specify the basic retrieval requests.
2. Expressive power of RA and RC are equivalent. This means any query that could be expressed in RA could be expressed by formula in RC.

3. (a) Assume a database about Company

EMPLOYEE (ss#, name)
COMPANY (cname, address)
WORKS (ss#, cname)
SUPERVISES (supervisor_ss#, employee_ss#)

Write relational algebra and SQL queries for each of the following cases.

a. Find the names of supervisors that work in companies whose address equals '**Biratnagar**'

```
SELECT name
FROM Employee e, Supervise s, Works w, Company c
WHERE C.address='Biratnagar' AND e.ss#=s.supervisor_ss# AND
s.supervisor_ss# = w.ss# AND w.cname=c.cname;
```

b. Find names of all companies who have more than 10 employees

```
SELECT cname, COUNT (employee_ss#) AS noe
FROM Company c, Works w, Superviser s
WHERE C.cname=w.cname AND w.ss#=s.Supervisor_ss#
GROUP BY cname
HAVING noe>10;
```

c. Find the name of supervisor who has the minimum number of employees.

```
SELECT name, MIN (noe)
FROM (SELECT name, supervisor_ss#, COUNT (employee_ss#) AS noe
        FROM Employee e, Supervise s
            WHERE e.ss#=s.supervisor_ss#
                GROUP BY supervisor_ss#);
```

(b) What is constraint? How does SQL allow implementation of general integrity constraints?

Solution: What is Constraints?

Constraints are the rules that are used to control data in columns of a particular relation.

Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database. Constraints are basically used to impose rules on the table, whenever a row is inserted, updated, or deleted from the table. Constraints prevent the deletion of a table if there are dependencies. The different types of constraints that can be imposed on the table are domain constraints, referential constraints, trigger, assertions etc. The constraints related to domain constraints are NOT NULL, UNIQUE, PRIMARY KEY and CHECK constraints.

1. Domain constraints
   - NOT NULL constraints
   - UNIQUE constraints
   - PRIMARY KEY constraints
   - CHECK constraints etc.
2. Referential constraints
3. Triggers
4. Assertion

Primary Key Constraint

When an attribute or set of attributes is declared as the primary key, then the attribute will not accept NULL value moreover it will not accept duplicate values. It is to be noted that "only one primary key can be defined for each table."

Syntax of primary key constraint is,

CREATE TABLE <table name**>**

(

Column name1, data-type of the column1 PRIMARY KEY,

Column name2, data-type of the column2,

Column nameN, data-type of the column

);

The above syntax indicates that column1 is declared as PRIMARY KEY constraint.

Example

CREATE TABLE Student

(

Sid INTEGER PRIMARY KEY,

sname VARCHAR(20),

age INTEGER UNIQUE

)

If we execute the following query,

INSERT INTO Student VALUES (1, "Abin", 04);

INSERT INTO Student VALUES (2, "Aayan", 11);

INSERT INTO Student VALUES (3, "Bindu", 26); then three records are successfully inserted into the student table and if we display such a table then we get following table,

sid	Sname	Age
1	Abin	04
2	Aayan	11
3	Bindu	26

But if we insert the duplicate aged record like below

INSERT INTO Student VALUES (4, "Umesh", 11); then we get duplicate entry for primary key attribute error message and insertion is failed. Also if we leave primary keys value for a particular record like below,

INSERT INTO Student VALUES ("Geeta", 25); then we get primary key cannot be null error message and insertion is failed.

Describe other constraints in brief itself.

4. (a). Define first, second and third normal form with suitable example.

Solution:

First Normal Form (1NF)

A relation is said to be in 1NF if and only if all domains of the relation contains only atomic (indivisible) values.

More simply a relation is in 1 NF if it does not have multi-valued attributes, composite attributes and their combinations. It states that the domain of an attribute must include only atomic (simple, indivisible) values.

Example: let's take an un-normalized relation containing composite attributes as,

Student

Sid	Sname	Subjects	
		Subject1	Subject2
1	Nitesh	DBMS	Graphics
2	Laxman	C	C++
3	Geeta	JAVA	.NET
4	Anisha	Simulation	SAD
5	Monika	Algebra	Calculus

The above table cannot be considered as an example of 1 NF, because it has repeating groups (two subject fields).Now convert this relation into 1 NF as,

Student

Sid	Sname	Subject1	Subject2
1	Nitesh	DBMS	Graphics
2	Laxman	C	C++
3	Geeta	JAVA	.NET
4	Anisha	Simulation	SAD
5	Monika	Algebra	Calculus

Fig: table in 1 NF

Example 2: let's take a relation that is not in 1 NF (containing multi-valued attributes)

Student

Sid	Sname	Address	Phone_No
1	Nitesh	Kalanki	9849145464 9813335467
2	Laxman	Balkhu	9841882345 099392844
3	Geeta	Kirtipur	9848334898
4	Anisha	Pokhara	9849283847
5	Monika	Ratopool	9840084732

			9803267499

Now converting this relation into 1 NF by decomposing this relation into two relations as,

Student

Sid	Sname	Address
1	Nitesh	Kalanki
2	Laxman	Balkhu
3	Geeta	Kirtipur
4	Anisha	Pokhara
5	Monika	Ratopool

Phone

Sid	Phone_No
1	9849145464
1	9813335467
2	9841882345
2	099392844
3	9848334898
4	9849283847
5	9840084732
5	9803267499

Fig: Relations in 1 NF

Second Normal Form (2NF)

A relation is said to be in 2NF if and only if

- ☞ It is already in 1NF and
- ☞ Every non-prime attribute is fully dependent on the primary key of the relation.

Speaking inversely, if a table has some attributes which is partially dependant on the primary key of that table then it is not in 2NF.

Example: let's take a relation in 1 NF but not in 2 NF as

Employee-Department

Emp-Id	Emp-Name	Emp-Salary	Dept-No	Dept-Name
1	Bhupi	40000	D1	BBA
1	Bhupi	40000	D2	CSIT
2	Bindu	30000	D3	BBS
3	Arjun	60000	D1	CSIT

In the above relation {Emp-Id, Dept-No} is the primary key. Emp-Name, Emp-Salary and Dept-Name all depend upon {Emp-Id, Dept-No}. Again Emp-Id→Emp-Name, Emp-Id→Emp-Salary and Dept-No→Dept-Name, thus there also occur partial dependency. Due to which this relation is not in 2 NF.

Now converting this relation into 2 NF by decomposing this relation into three relations as,

Employee ***Emp-Dept*** ***Department***

Emp-Id	**Emp-**	**Emp-**

	Name	Salary
1	Bhupi	40000
2	Bindu	30000
3	Arjun	60000

Emp-Id	*Dept-No*
1	*D1*
1	*D2*
2	*D3*
3	*D1*

Dept-No	*Dept-Name*
D1	BBA
D2	CSIT
D3	BBS

Fig: Relations in 2 NF

Third Normal Form (3NF)

A relation is said to be in 3NF if and only if

- ☞ it is already in 2NF and
- ☞ Every non-prime attribute is **non-transitively** dependent on the primary key.

Speaking inversely, if a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.

Example:

Student

S-Id	S-Name	Age	Sex	Hostel-Name
1	Laxmi	21	Female	White house
2	Binita	22	Female	White house
3	Rajesh	32	Male	Red house
4	Aayan	21	Male	Red house

Here S-Id→S-Name, S-Id→Age, S-Id→Sex, S-Id→Hostel-Name and also Sex→Hostel-Name

This last dependency was not originally specified but we have derived it. The derived dependency is called a transitive dependency**.** In such a case the relation should be broken into two relations as,

Student ***Hostel***

S-Id	S-Name	Age	Sex
1	Laxmi	21	Female
2	Binita	22	Female
3	Rajesh	32	Male
4	Aayan	21	Male

Sex	Hostel-Name
Female	White house
Male	Red house

Fig: Relations in 3 NF

(b) What is functional dependency? Describe full and partial functional dependency with suitable example.

Solution: Functional dependencies are the relationships among the attributes within a relation. Functional dependencies provide a formal mechanism to express constraints between attributes. If attribute A functionally depends on attribute B, then for every instance of B you will know the respective value of A. Attribute "B" is functionally

dependent upon attribute "A" (or collection of attributes) if a value of "A" determines or single value of attributes "B" at only one time functional dependency helps to identify how attributes are related to each other.

Let A and B are attributes of a relation R. If each value of B is associated with exactly one value of A then B is said to be functionally dependent on A. it is denoted by A→B.

Example

Student (sid, sname, address, age)

Here, sname, address and age are functionally dependent on sid. Meaning is that each student id uniquely determines the value of attributes student name, address and age. This can be express by,

sid → sname

sid → address

sid → age

Types of functional dependency

There are many types of functional dependencies, depending on several criteria

- Fully functional dependency
- Partial functional dependency
- Trivial and non-trivial dependency
- Transitive dependency

Fully functional dependency

An attribute is fully functionally dependent on a second attribute if and only if it is functionally dependent on the second attribute but not on any subset of the second attribute.

Mathematically, for a relation schema R, in a functional dependency X→Y, Y is said to be fully functional dependent on X if Z→Y is false for all $Z \subset X$.

Partial functional dependency

An attribute is partial functionally dependent on a second attribute if and only if it is functionally dependent on the second attribute and also dependency occur on any subset of the second attribute. This is the situation that exists if it is necessary to only use a subset of the attributes of the composite determinant to identify its object.

Mathematically, for a relation schema R, in a functional dependency X→Y, Y is said to be partial functional dependent on X if by removal of some attributes from X and the dependency still holds.

Example

The dependency {Emp-Id, Projecta-No}→Emp-Name is partial because Emp-Id→Emp-Name also holds.

5. (a) Draw a state diagram, and discuss the typical state that a transaction goes through during transaction.

Solution: Whenever a transaction is submitted to a DBMS for execution, either it executes successfully or fails due to some reasons. During its execution, a transaction passes through various states that are *active*, *partially committed*, *committed*, *failed*, and *aborted*. Figure 10.1 shows the state transition diagram that describes how a transaction passes through various states during its execution.

- **Active State:** In this state the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed State:** When a transaction executes its final operation, it is said to be in this state. After execution of all operations, the database system performs some checks e.g. the consistency state of database after applying output of transaction onto the database.
- **Failed State:** If any check made by database recovery system fails, the transaction is said to be in failed state, from where it can no longer proceed further.
- **Aborted:** If any of checks fails and transaction reached in failed state, the recovery manager rolls back all its write operation on the database to make database in the state where it was prior to start of execution of transaction. Transactions in this state are called aborted. Database recovery module can select one of the two operations after a transaction aborts:
  - Re-start the transaction
  - Kill the transaction
- **Committed State:** If transaction executes all its operations successfully it is said to be committed. All its effects are now permanently made on database system.

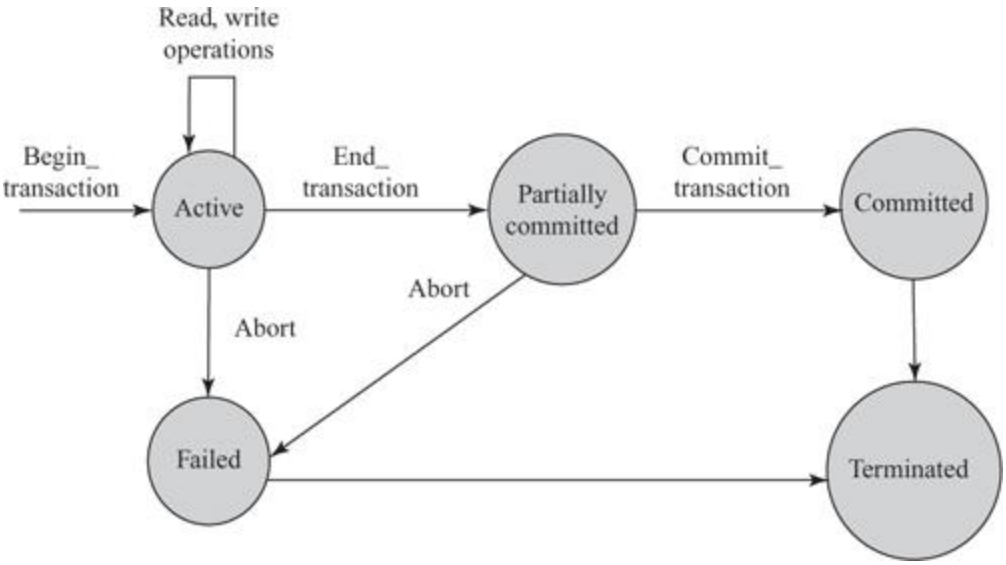


Fig: State diagram of transaction

(b). Describe serial and serializable schedule? Why serializable schedule is considered correct?

Solution: A schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle then next transaction is executed. Transactions are ordered one after other. This type of schedule is called *serial schedule.*

Example:

T1	T2
Read(A) A= A-50 Write(A) Read(B) B=B+50 Write(B)	
	Read(A) Temp= A*0.1 A=A-Temp Write(A) Read(B) B= B + Temp Write(B)

A given non serial schedule of n transactions is serializable if it is equivalent to some serial schedule. That is if a non-serial schedule produce the same result as of the serial schedule then the given non-serial schedule is said to be serializable.

Example:

T1	T2
Read(x)	
	Read(x)
Write(y)	
Commit	
	Write(x)
	Commit

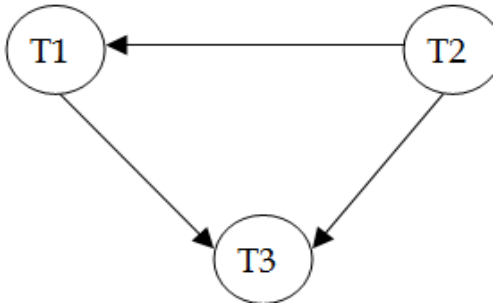
A serial schedule is considered correct and serializable schedule is also considered correct only if it is equivalent to some serial schedule.

Example:

Schedule: $R_2(x)$, $W_2(x)$, $R_1(x)$, $W_1(x)$, $R_2(y)$, $R3(x)$, $W_2(y)$, $W3(x)$

Solution:

Precedence graph for above schedule is given below



Since, above precedence graph of the schedule H, with 3 transactions doe not contains, the given schedule H is conflict serializable. The equivalent serial schedule can be achieved if transaction operations are taken in order T2→T1→T3. Thus equivalent serial schedule is: $R_2(x)$, $W_2(x)$, $R_2(y)$, $W_2(y)$, $R_1(x)$, $W_1(x)$, $R3(x)$, $W3(x)$.

6. (a). How does the granularity of data items affect the performance of concurrency control? What factors affect selection of granularity size for data items?

Solution:

In all concurrency control schemes, we have used each individual data item as the unit on which synchronization is performed. However, it would be advantageous to group

several data items and to treat them as one individual unit. For example, if a transaction Ti needs to access the entire database and it uses a locking protocol, then Ti must lock each item in the database, which is time consuming process. Hence it would be better if Ti would issue a single lock request to lock the entire database. On the other hand if transaction Ti needs to access only a few data items, it should not be required to lock the entire database. A data item can be one of the following.

2. Database Field
3. Database record
4. Disk block
5. Whole file
6. Whole database

The size of database item is often called the data item granularity. Fine granularity refers to overall item size where as coarse granularity refers to large item sizes. The best item size depends on the type transaction. Hierarchy of data granularities, where the small granularities are nested within larger ones, can be represented graphically a tree. In the tree, each node represents independent data item, non-leaf node of the multiple granularity tree represents the data associated with its descendents.

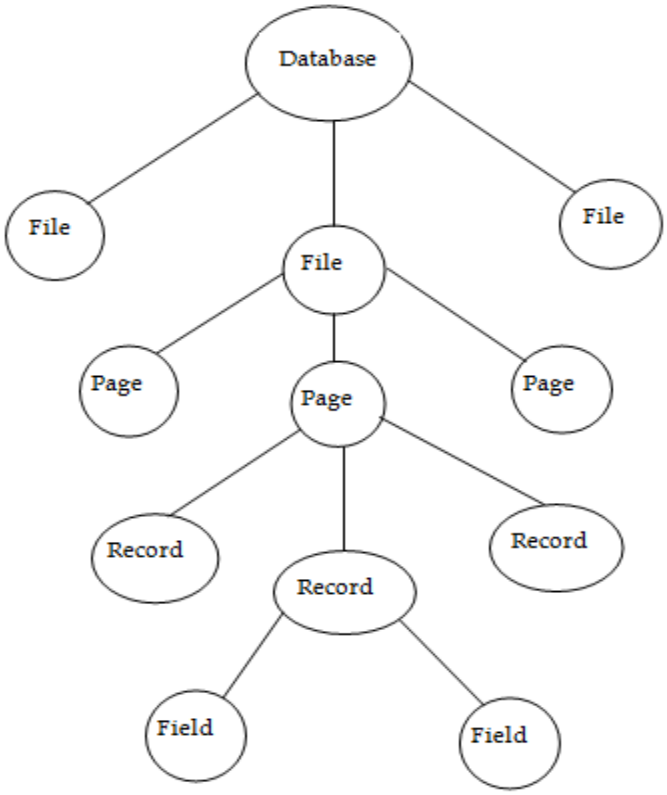


Figure: Hierarchy of Data Granurality

(b). Describe Write-ahead logging protocol.

Solution: Write-ahead log (WAL) protocol guarantees that no data modifications are written to disk before the associated log record is written to disk. This maintains the ACID properties for a transaction. At the time a modification is made to a page in the buffer, a log record is built in the log cache that records the modification. This log record must be written to disk before the associated dirty page is flushed from the buffer cache to disk. If the dirty page is flushed before the log record is written, the dirty page creates a modification on the disk that cannot be rolled back if the transaction fails before the log record is written to disk. WAL states that

- For Undo: Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
- For Redo: Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

TU 2070

1 (a) what is database management system? Discuss the advantages of using database management system over file system.

Solution: A **Database Management System (DBMS)** is the set of programs that is used to store, retrieve and manipulate the data in convenient and efficient way. Main goal of database management system (DBMS) is to hide underlying complexities of data management from users and provide easy interface to them. Some common examples of the DBMS software are Oracle, Sybase, Microsoft SQL Server, DB2, MySQL, Dbase, Ms-Access etc.

The benefits (Advantages) of using DBMS are:

- **To reduce redundancy:** Repeating of the same information in database is called redundancy of data which leads to several problems such as wastage of space, duplication effort for entering data and inconsistency. When DBMS is used and database is created, redundancy is minimized.
- **To avoid inconsistency:** The database is said to be inconsistent if various copies of the same data may no longer agree. For example, a changed customer address may be reflected in saving account but not elsewhere in the system. By using DBMS we can avoid inconsistency.

- **To share data:** The data in the database can be shared among many users and applications. The data requirements of new applications may be satisfied without having to create any new stored files.
- **To provide support for transactions:** A transaction is a sequence of database operations that represents a logical unit of work. It accesses a database and transforms it from one state to another. A transaction can update a record, delete one, modify a set of records etc. when the DBMS does a 'commit', the changes made by the transaction are made permanent. We can roll back the transaction to undo the effects of transaction.
- **To maintain integrity:** Most database applications have certain integrity constraints that must hold for the data. A DBMS provides capabilities for defining and enforcing these constraints. For example, the value of roll number field of each student in student database should be unique for each student. It is a type of rule. Such a rule is enforced using constraint at the time of creation of database.
- **To enforce security:** Not every user of the database system should be able to access all data. Different checks can be established for each type of access (retrieve, modify, delete, etc) to each piece of information in the database.
- **To provide efficient backup and recovery:** Provide facilities for recovering from software and hardware failures to restore database to previous consistent state.
- **To Concurrent Access Database:** Concurrent access means access to the same data simultaneously by more than one user. The same data may be used by many users for the purpose reading at the same time. But when a user tries to modify a data, there should be a concurrency control mechanism to avoid the inconsistency of data. A DBMS provides facilities for these operations.

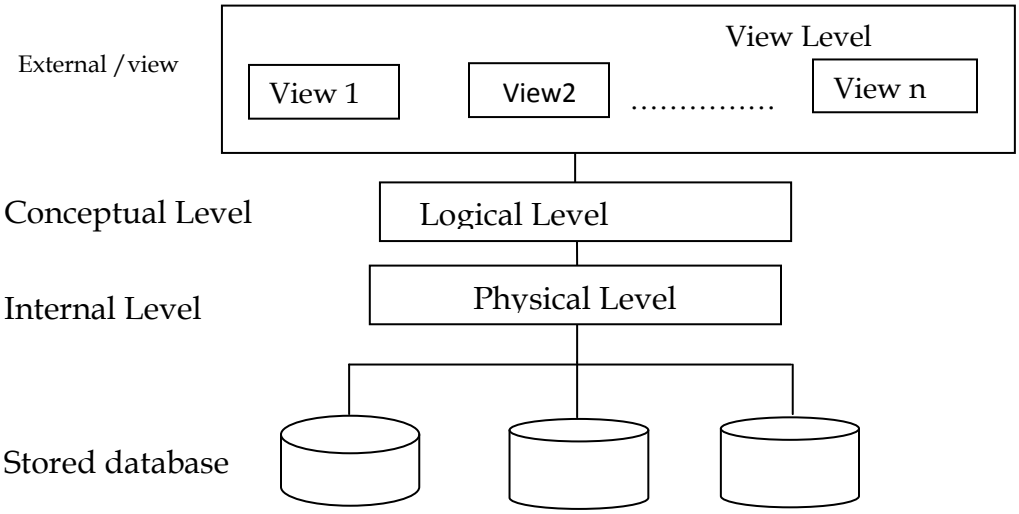
(b) What is data abstraction? Discuss three levels of this abstraction.

Solution: Data abstraction is the process of easy interface to users by hiding underlying complexities of data management from users. Database System provides users with an *abstract view* of the data. Data abstraction is provided in database management systems by using three-level schema (ANSI /SPARC) architecture.

Physical level

It is the lowest level of abstractions and describes **how** the data are actually stored on disk and some of the access mechanisms commonly used for retrieving this data. While designing this layer, the main objective is to optimize performance by minimizing the

number of disk accesses during the various database operations. The database system hides many of the lowest level storage details from database programmer**. DBMS developer** is the person who deals with this level. Database administrator may be aware of certain details of the physical organization of the data.



Logical level

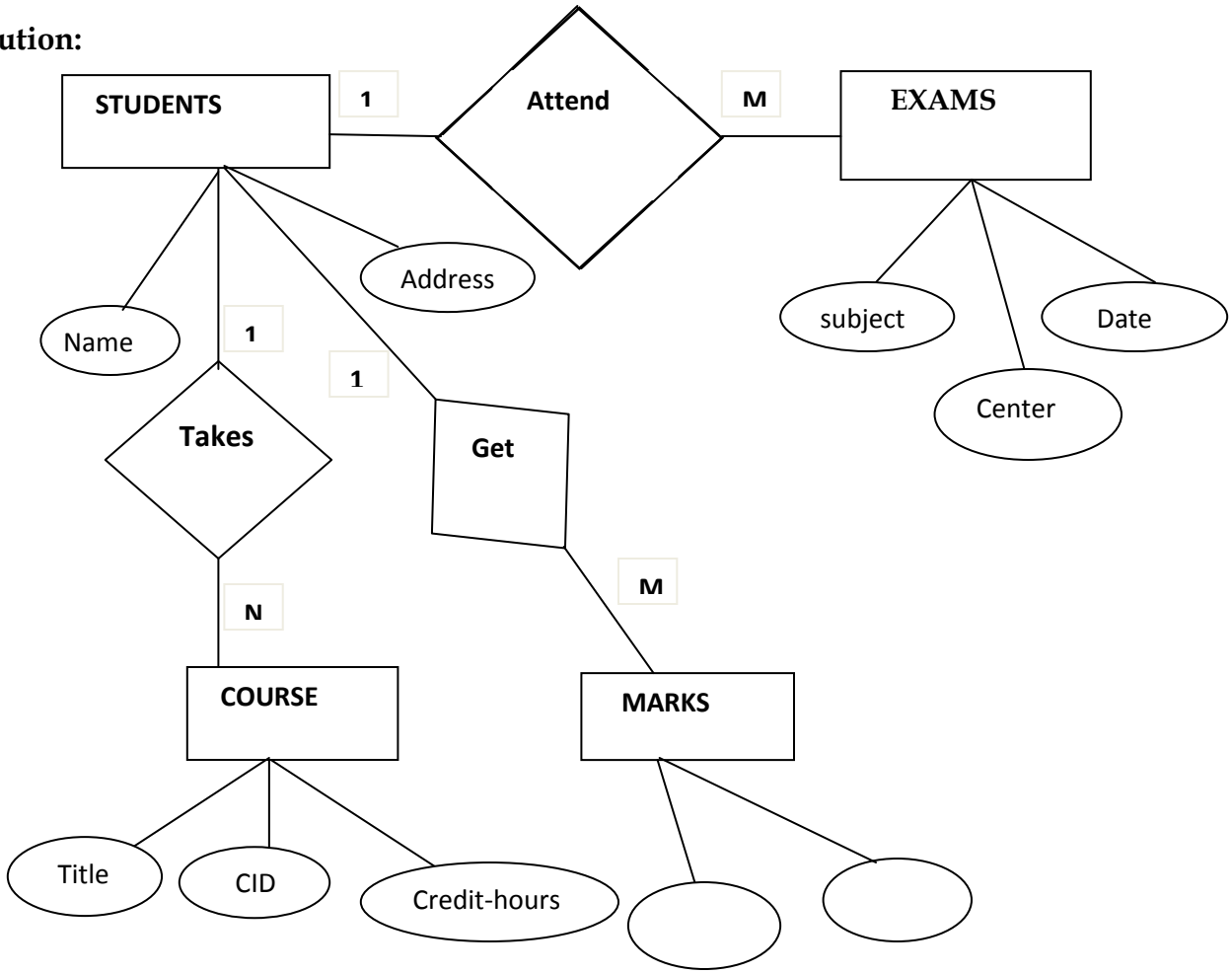
It is the next higher level of data abstraction which describes **what** data are stored in the database, and what relationships exist among those data. It is also known as conceptual level at the schema at this level is called **logical schema** (conceptual schema). Logical level describes the stored data in terms of the data model of the DBMS. Programmers and database administrator works at this level of abstraction. Database users do not need to have knowledge of this level.

View level

It is the highest level of abstraction and describes only a **part of the database** and hides some information from the user. At view level, computer users see a set of application programs that hide details of data types. Similarly at the view level **several views** of the database are defined and database user see only these views. Schema at this level is called **external schema**. Views also plays vital role to provide **security** mechanism to prevent users from accessing certain parts of the database such as an employee's salary for security purposes.

2 (a) Construct an ER diagram (ERD) to record the marks that students get in different exams of different course offerings.

Solution:



(b) Define integrity constraint? Discuss domain constraint with suitable example.

Solution:

Domain constraint

Domain constraints define the set of values permitted in attribute of a relation. In simplest form domain constraints can be applied to the database columns by defining their data types. Domain constraints are the most elementary form of integrity constraint. They are tested essentially by the system whenever a new data item is entered into the database. Domain integrity means the definition of a valid set of values for an attribute. We can define

- ✓ data type
- ✓ Length or size
- ✓ Is null value allowed

- ✓ Is the value unique or not for an attribute etc.

Student

A1	A2	……	An

In the above student table, the attribute A1 draws value from domain D1, A2 from D2 and so on.

SQL allows us to create new domains from existing data types by using **create domain** clause as below:

create domain *Dollars* **numeric**(12, 2)

create domain *Pounds* **numeric**(12,2)

Once the domains are created we can use them as data types of attributes while creating relations as below:

Create table employee

(

eid **varchar(5),**

ename **varchar(10),**

Salary **dollars**

)

3. (a) With the information given below, calculate any three members of F+

R=(A, B, C, G, H, I)

F={A→B, A→C, CG→I, B→H}

Compute closure of F (F^+).

Solution:

Since A→B, B→H then A→H [by transitivity rule]

Since A→B, A→C then A→BC [by union rule]

Since A→C then AG→CG [by augmentation rule]

Since AG→CG, CG→I then AG→I [by transitivity rule]

Since CG→I then C→I, G→I [By decomposition rule]

Hence, F^+ = {A→A, B→B, C→C, H→H, G→G, I→I, A→B, A→C, C→I, G→I, B→H, A→H, AG→I, AG→CG}

Here, first six FDs obtain by reflexive axiom.

(b). Discuss 2NF and 3NF with suitable example.

Solution:

Second Normal Form (2NF)

A relation is said to be in 2NF if and only if

- ☞ It is already in 1NF and
- ☞ Every non-prime attribute is fully dependent on the primary key of the relation.

Speaking inversely, if a table has some attributes which is partially dependant on the primary key of that table then it is not in 2NF.
Example: let's take a relation in 1 NF but not in 2 NF as

Employee-Department

Emp-Id	Emp-Name	Emp-Salary	Dept-No	Dept-Name
1	Bhupi	40000	D1	BBA
1	Bhupi	40000	D2	CSIT
2	Bindu	30000	D3	BBS
3	Arjun	60000	D1	CSIT

In the above relation {Emp-Id, Dept-No} is the primary key. Emp-Name, Emp-Salary and Dept-Name all depend upon {Emp-Id, Dept-No}. Again Emp-Id→Emp-Name, Emp-Id→Emp-Salary and Dept-No→Dept-Name, thus there also occur partial dependency. Due to which this relation is not in 2 NF.
Now converting this relation into 2 NF by decomposing this relation into three relations as,

Employee

Emp-Id	**Emp-Name**	**Emp-Salary**
1	Bhupi	40000
2	Bindu	30000
3	Arjun	60000

Emp-Dept

Emp-Id	*Dept-No*
1	*D1*
1	*D2*
2	*D3*
3	*D1*

Department

Dept-No	*Dept-Name*
D1	BBA
D2	CSIT
D3	BBS

Fig: Relations in 2 NF

Third Normal Form (3NF)

A relation is said to be in 3NF if and only if

- ☞ it is already in 2NF and
- ☞ Every non-prime attribute is **non-transitively** dependent on the primary key.

Speaking inversely, if a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.

Example:

Student

S-Id	S-Name	Age	Sex	Hostel-Name
1	Laxmi	21	Female	White house
2	Binita	22	Female	White house
3	Rajesh	32	Male	Red house
4	Aayan	21	Male	Red house

Here S-Id→S-Name, S-Id→Age, S-Id→Sex, S-Id→Hostel-Name and also Sex→Hostel-Name

This last dependency was not originally specified but we have derived it. The derived dependency is called a transitive dependency**.** In such a case the relation should be broken into two relations as,

Student

S-Id	S-Name	Age	Sex
1	Laxmi	21	Female
2	Binita	22	Female
3	Rajesh	32	Male
4	Aayan	21	Male

Hostel

Sex	Hostel-Name
Female	White house
Male	Red house

Fig: Relations in 3 NF

4. Consider the following supplier database, where primary keys are underlined.

supplier(supplier-id, supplier-name, city)

supplies(supplier-id, part-id, quantity)

parts(part-id, part-name, color, weight)

Construct the following relational algebra queries for this relational database

a Find the name of all suppliers located in city "Kathmandu".

$$\Pi_{\text{supplier-name}}(\sigma_{\text{city="Kathmandu"}}(\text{supplier}))$$

b Find the name of all parts supplied "ABC Company".

$$\Pi_{\text{part-name}}(\sigma_{\text{supplier-name="ABC Company"}}(\text{supplier} \bowtie \text{supplies} \bowtie \text{parts}))$$

c Find the name of all parts that are supplied in quantity greater than 300.

$$\Pi_{\text{part-name}}(\sigma_{\text{quantity>300}}(\text{supplies} \bowtie \text{parts}))$$

d Find the number of parts supplied by "ABC Company".

$$G_{\text{COUNT (Part-id)}}(\sigma_{\text{supplier-name="ABC Company"}}(\text{supplier} \bowtie \text{supplies} \bowtie \text{parts}))$$

e Find the name of all suppliers who supply more than 30 different parts.

$$\text{supplier-name}\, G_{\text{COUNT(part-id)}>30}(\text{supplier} \bowtie \text{supplies} \bowtie \text{parts})$$

5. (a). What is serializable schedule? How can you test a schedule for conflict serializability?

Solution: A given non serial schedule of n transactions are serializable if it is equivalent to some serial schedule.That is if a non-serial schedule produce the same result as of the serial schedule then the given non-serial schedule is said to be serializable.A schedule that is not serializable is called a non-serializable. The main objective of serializability is to search non-serial schedules that allow transaction to execute concurrently without interfering one another transaction and produce the result that could be produced by a serial execution. In serializability, ordering of read/write is important. If two transactions only read data item they do not conflict and order is not important. If two transactions either read or write completely separate data items, they do not conflict and order is not important. If one transaction writes a data item and another reads or writes same data items, then order of execution is important.

Testing for Conflict Serializability of a Schedule

Schedule compliance with conflict serializability can be tested with the precedence graph (serializability graph, serialization graph, conflict graph). Precedence graph is the

directed graph representing precedence of transactions in the schedule, as reflected by precedence of conflicting operations in the transactions. In the precedence graph transactions are nodes and precedence relations are directed edges. There exists an edge from a first transaction to a second transaction, if the second transaction is in conflict with the first. A schedule is conflict-serializable if and only if its precedence graph is acyclic. This means that a cycle consisting of committed transactions only is generated in the precedence graph, if and only if conflict-serializability is violated.

There is a simple algorithm for determining whether a particular schedule is conflict serializable or not. If precedence graph is acyclic, the serializability order can be obtained by a **topological sorting**of the graph.

Algorithm Testing Conflict Serializability of a Schedule *S*

1 For each transaction *Ti* participating in schedule *S,* create a node labeled *Ti* in the precedence graph.

2 For each case in *S* where *Tj* executes a Read(x) operation after *Ti* executes a write(x) operation, create an edge (*Ti*→ *Tj*) in the precedence graph.

3 For each case in *S* where *Tj* executes a write(x) operation after *Ti* executes a read(x) operation, create an edge (*Ti*→*Tj*) in the precedence graph.

4 For each case in *S* where *Tj* executes a write(x) operation after *Ti* executes a write(x), create an edge (*Ti*→ *Tj*) in the precedence graph.

5 Test the precedence graph for the existence of cycle. If cycle is encountered, schedule is not serializale. Otherwise schedule is serializable.

Example 1: Draw precedence graph for following schedule and identify wheather it is conflict serizable or not. If it is conflict serializable, identify equivalent serial schedule also.

T1	T2	T3
Read(x) Write(x) Commit	 Write(x) Commit	 Write(x) Commit

Solution

Precedence graph for above schedule is given below

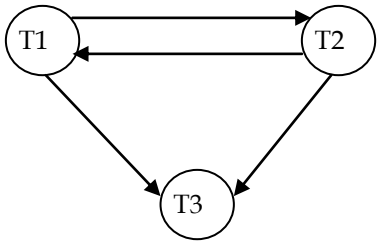


Figure: Precedence Graph for Schedule G

Since, above precedence graph of the schedule G, with 3 transactions contains a cycle (of length 2; with two edges) through, the given schedule G is not conflict serializable.

(b). Discuss recovery technique base on deferred update with concurrent execution in multiuser environment.

Solution: *Recovery Using Deferred Update in a Multiuser environment* is closely interrelated with concurrency control. This environment requires some concurrency control mechanism to guarantee isolation property of transactions. It also uses two lists of transactions: committed transactions since the last checkpoint (commit list) and 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 and the transactions that are active and did not commit are cancelled and must be resubmitted. Consider a schedule consisting of concurrent transactions and associated log file given below:

T1	T2	T3	T4
Read(w) Write(w) Read(x) Write(x) Commit			Read(y) Write(y) Read(w) Write(w) Commit
	Read(y) Write(y) Read(x) Write(x) Commit	Read(w) Write(w) Read(z) Write(z) Commit	

Schedule R3

Tid	Operation	DataItem	OldValue	NewValue
T1	Start			
T1	Write	w	400	300
T1	Write	x		
T1	Commit			
Checkpont				
T4	Start			
T4	Write	y	200	350
T4	Write	w	500	250
T4	Commit			
T2	Start			
T2	Write	y		
T3	Start			
T3	Write	w		
T2	Write	x		

Log File Recored for Schedule

Syste

In the above example transactions T2 and T3 are ignored because they did not reach their commit point but transaction T4 is redone because its commit point is after the last checkpoint. On the other hand we do not need to redo transaction T1 because its commit point is before last checkpoint and hence effect of T1 is already recorded in database. For example, in case schedule R2 given above transaction T1 is redone because it is in committed list and transaction T2 is undone becaused transaction is failed before reaching to commit point.

The above 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. For example, in case of schedule R3 given above transactions T2 and T3 are undone because they are in active list but transaction T4 is redone because it committed after last checkpoint. But recovery manager ignores transaction T1 because its commit point is before last checkpoint.