2017-18 SECTION A AND B SOLVED

SECTION A

Q1

a- Explain Specialization

Ans- Specialization is a method of production whereby an entity focuses on the production of a limited scope of goods to gain a greater degree of efficiency. Many countries, for example, specialize in producing the goods and services that are native to their part of the world, and they trade for other goods and services.Specialization is an agreement within a community, organization, or larger group where each of the members best suited for a specific activity assumes responsibility for its successful execution.

b- Write advantages of Database

Ans-A Database Management System (DBMS) is defined as the software system that allows users to define, create, maintain and control access to the database. DBMS makes it possible for end users to create, read, update and delete data in database. It is a layer between programs and data.

Compared to the File Based Data Management System, Database Management System has many advantages. Some of these advantages are given below:

Reducing Data Redundancy

The file based data management systems contained multiple files that were stored in many different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy.

This is prevented in a database as there is a single database and any change in it is reflected immediately. Because of this, there is no chance of encountering duplicate data.

Sharing of Data

In a database, the users of the database can share the data among themselves. There are various levels of authorisation to access the data, and consequently the data can only be shared based on the correct authorisation protocols being followed.

Many remote users can also access the database simultaneously and share the data between themselves.

Data Integrity

Data integrity means that the data is accurate and consistent in the database. Data Integrity is very important as there are multiple databases in a DBMS. All of these databases contain data that is visible to multiple users. So it is necessary to ensure that the data is correct and consistent in all the databases and for all the users.

Data Security

Data Security is vital concept in a database. Only authorised users should be allowed to access the database and their identity should be authenticated using a username and password. Unauthorised users should not be allowed to access the database under any circumstances as it violates the integrity constraints.

Privacy

The privacy rule in a database means only the authorized users can access a database according to its privacy constraints. There are levels of database access and a user can only view the data he is allowed to. For example - In social networking sites, access constraints are different for different accounts a user may want to access.

Backup and Recovery

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS. Moreover, it also restores the database after a crash or system failure to its previous condition.

Data Consistency

Data consistency is ensured in a database because there is no data redundancy. All data appears consistently across the database and the data is same for all the users viewing the database. Moreover, any changes made to the database are immediately reflected to all the users and there is no data inconsistency.

c- Define DML

Ans- SQL is equipped with data manipulation language (DML). DML modifies the database instance by inserting, updating and deleting its data. DML is responsible for all forms data modification in a database. SQL contains the following set of commands in its DML section −

- SELECT/FROM/WHERE
- INSERT INTO/VALUES
- UPDATE/SET/WHERE
- DELETE FROM/WHERE

These basic constructs allow database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

SELECT/FROM/WHERE

- **SELECT** − This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.
- **FROM** − This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.
- **WHERE** − This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.

For example −

```
Select author_name
From book_author
Where age > 50;
```

This command will yield the names of authors from the relation **book_author** whose age is greater than 50.

INSERT INTO/VALUES

This command is used for inserting values into the rows of a table (relation).

Syntax−

INSERT INTO table (column1 [, column2, column3 ...]) VALUES (value1 [, value2, value3 ...])

Or

INSERT INTO table VALUES (value1, [value2, ...])

For example −

INSERT INTO book (Author, Subject) VALUES ("anonymous", "computers");

UPDATE/SET/WHERE

This command is used for updating or modifying the values of columns in a table (relation).

Syntax −

UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition]

For example −

UPDATE book SET Author="webmaster" WHERE Author="anonymous";

DELETE/FROM/WHERE

This command is used for removing one or more rows from a table (relation).

Syntax −

DELETE FROM table_name [WHERE condition];

For example −

DELETE FROM book
WHERE Author="unknown";

d- Explain logical data independency

Ans- Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints, applied on that relation.

Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk.

e- Explain entity integrity constraints

- Ans- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

f- Define 2NF

Ans- The second step in Normalization is 2NF.

A table is in 2NF, only if a relation is in 1NF and meet all the rules, and every non-key attribute is fully dependent on primary key.

The Second Normal Form eliminates partial dependencies on primary keys.

Let us see an example:

Example (Table violates 2NF)

<StudentProject>

StudentID	ProjectID	StudentName	ProjectName
S89	P09	Olivia	Geo Location
S76	P07	Jacob	Cluster Exploration
S56	P03	Ava	IoT Devices
S92	P05	Alexandra	Cloud Deployment

In the above table, we have partial dependency; let us see how:

The prime key attributes are **StudentID** and **ProjectID**.

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent.

The **StudentName** can be determined by **StudentID**, which makes the relation Partial Dependent.

The **ProjectName** can be determined by **ProjectID**, which makes the relation Partial Dependent.

Therefore, the <**StudentProject**> relation violates the 2NF in Normalization and is considered a bad database design.

Example (Table converted to 2NF)

To remove Partial Dependency and violation on 2NF, decompose the above tables:

<StudentInfo>

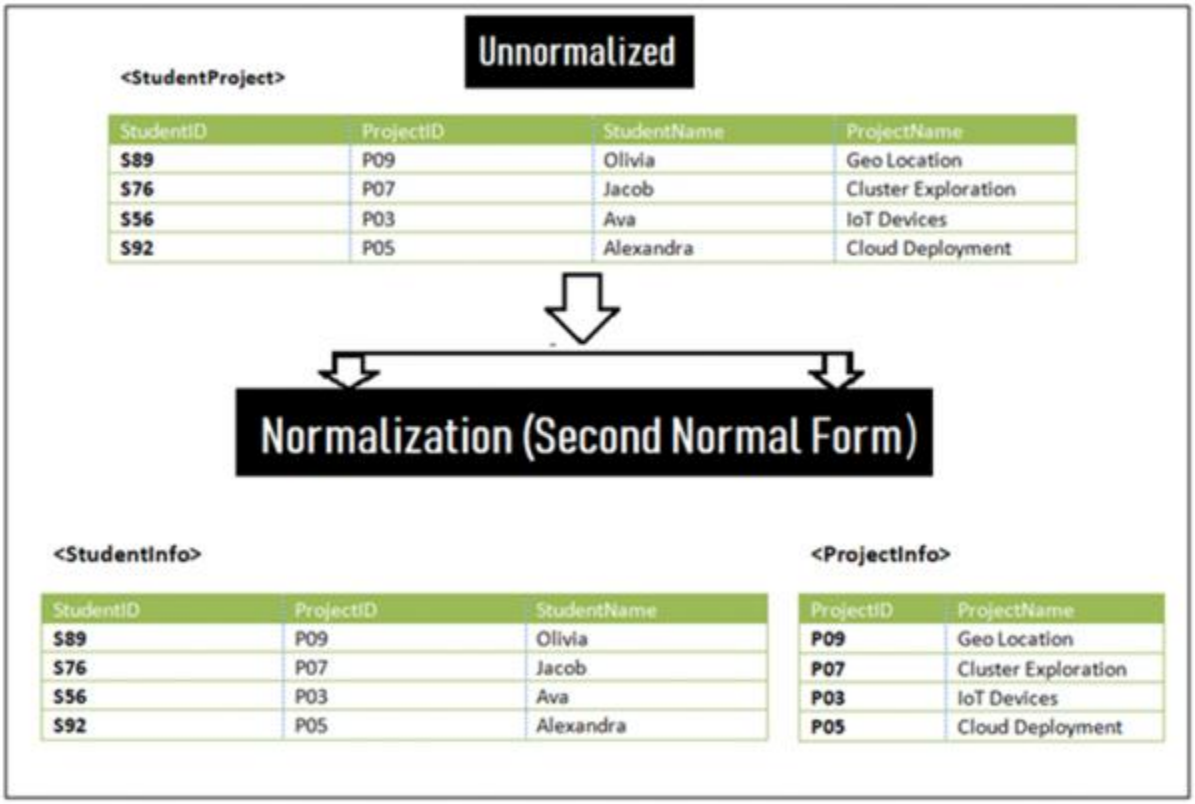
StudentID	ProjectID	StudentName
S89	P09	Olivia
S76	P07	Jacob
S56	P03	Ava
S92	P05	Alexandra

<ProjectInfo>

ProjectID	ProjectName
P09	Geo Location

P07	Cluster Exploration
P03	IoT Devices
P05	Cloud Deployment

Now the relation is in 2nd Normal form of Database Normalization



g- Explain I in ACID property
Ans- In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

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

i- Define exclusive lock

Ans-When a statement modifies data, its transaction holds an *exclusive* lock on data that prevents other transactions from accessing the data.

This lock remains in place until the transaction holding the lock issues a commit or rollback. Table-level locking lowers concurrency in a multi-user system.

J- Define replication in distributed database

Ans-**Data Replication** is the process of storing data in more than one site or node. It is useful in **improving the availability of data**. It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency. The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others.
Data replication encompasses duplication of transactions on an ongoing basis, so that the **replicate is in a consistently updated state** and synchronized with the source.However in data replication data is available at different locations, but a particular relation has to reside at only one location.

Section-B

a- Discuss the role of database administrator.

The role of the DBA is very important and is defined by the following functions

- **Defining the Schema** The DBA defines the schema which contains the structure of the data in the application. The DBA determines what data needs to be present in the system ad how this data has to be represented and organized. As per this construction, database will be produced to store required information for an association.
- **Defining Storage Structure and Access Method**: The DBA chooses how the information is to be spoken to in the put away database.
- **Liaising with Users** The DBA needs to interact continuously with the users to understand the data in the system and its use. The DBA figures out which client needs access to which part of the database
- **Defining Security & Integrity Checks** The DBA finds about the access restrictions to be defined and defines security checks accordingly. Data Integrity checks are also defined by the DBA.
- **Defining Backup / Recovery Procedures** The DBA also defines procedures for backup and recovery. Defining backup procedures includes specifying what data is to backed up, the periodicity of taking backups and also the medium and storage place for the backup data.
- **Monitoring Performance** The DBA has to continuously monitor the performance of the queries and take measures to optimize all the queries in the application.
- **Assistance to Application Programmers**: The DBA gives help to application software engineers to create application programs.

b- Discuss the join type with suitable example
Ans- SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

JOIN Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is **(n-1)** where **n**, is number of tables. A table can also join to itself, which is known as, **Self Join**.

Types of JOIN

Following are the types of JOIN that we can use in SQL:

- Inner
- Outer
- Left
- Right

Cross JOIN or Cartesian Product

This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

```
SELECTcolumn-name-list
FROM
table-name1CROSSJOINtable-name2;
```

Example of Cross JOIN

Following is the **class** table,

ID	NAME
1	Abhi
2	Adam
4	Alex

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Cross JOIN query will be,

```
SELECT*FROM
classCROSSJOINclass_info;
```

The resultset table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	1	DELHI
4	alex	1	DELHI
1	abhi	2	MUMBAI
2	adam	2	MUMBAI
4	alex	2	MUMBAI
1	abhi	3	CHENNAI
2	adam	3	CHENNAI
4	alex	3	CHENNAI

As you can see, this join returns the cross product of all the records present in both the tables.

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

```
SELECTcolumn-name-list FROM
table-name1INNERJOINtable-name2
WHEREtable-name1.column-name =table-name2.column-name;
```

Example of INNER JOIN

Consider a **class** table,

ID	NAME
1	Abhi
2	Adam
3	Alex
4	Anu

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Inner JOIN query will be,

```
SELECT*from class INNERJOINclass_infowhere class.id = class_info.id;
```

The resultset table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

The syntax for Natural Join is,

```
SELECT*FROM

table-name1NATURALJOINtable-name2;
```

Example of Natural JOIN

Here is the **class** table,

ID	NAME
1	Abhi
2	Adam
3	Alex
4	Anu

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Natural join query will be,

```
SELECT*from class NATURALJOINclass_info;
```

The resultset table will look like,

ID	NAME	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

In the above example, both the tables being joined have **ID** column(same name and same datatype), hence the records for which value of **ID** matches in both the tables will be the result of Natural Join of these two tables.

OUTER JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

LEFT Outer Join

The left outer join returns a resultset table with the **matched data** from the two tables and then the remaining rows of the **left** table and null from the **right** table's columns.

Syntax for Left Outer Join is,

```
SELECTcolumn-name-list FROM
table-name1LEFTOUTERJOINtable-name2
ONtable-name1.column-name =table-name2.column-name;
```

To specify a condition, we use the ON keyword with Outer Join.

Left outer Join Syntax for **Oracle** is,

```
SELECTcolumn-name-list FROM
table-name1,table-name2 ontable-name1.column-name =table-name2.column-name(+);
```

Example of Left Outer Join

Here is the **class** table,

ID	NAME
1	Abhi
2	Adam
3	Alex
4	Anu
5	Ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Left Outer Join query will be,

```
SELECT*FROM class LEFTOUTERJOINclass_infoON(class.id = class_info.id);
```

The resultset table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI

2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	Null
5	ashish	null	Null

RIGHT Outer Join

The right outer join returns a resultset table with the **matched data** from the two tables being joined, then the remaining rows of the **right** table and null for the remaining **left** table's columns.

Syntax for Right Outer Join is,

```
SELECTcolumn-name-list FROM
table-name1RIGHTOUTERJOINtable-name2
ONtable-name1.column-name =table-name2.column-name;
```

Right outer Join Syntax for **Oracle** is,

```
SELECTcolumn-name-list FROM
table-name1,table-name2
ONtable-name1.column-name(+)=table-name2.column-name;
```

Example of Right Outer Join

Once again the **class** table,

ID	**NAME**
1	Abhi
2	Adam
3	Alex

4	Anu
5	Ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Right Outer Join query will be,

```
SELECT*FROM class RIGHTOUTERJOINclass_infoON(class.id = class_info.id);
```

The resultant table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
null	null	7	NOIDA
null	null	8	PANIPAT

Full Outer Join

The full outer join returns a resultset table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

Syntax of Full Outer Join is,

```
SELECTcolumn-name-list FROM
table-name1FULLOUTERJOINtable-name2
ONtable-name1.column-name =table-name2.column-name;
```

Example of Full outer join is,

The **class** table,

ID	NAME
1	Abhi
2	Adam
3	Alex
4	Anu
5	Ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

7	NOIDA
8	PANIPAT

Full Outer Join query will be like,

```
SELECT*FROM class FULLOUTERJOINclass_infoON(class.id = class_info.id);
```

The resultset table will look like,

ID	**NAME**	**ID**	**Address**
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null
Null	null	7	NOIDA
Null	null	8	PANIPAT

---

c- What is trigger? Explain different trigger with example

Ans-Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

d- Write difference between BCNF VS 3NF?

Ans-

BASIS FOR COMPARISON	3NF	BCNF
Concept	No non-prime attribute must be transitively dependent on the Candidate key.	For any trivial dependency in a relation R say X->Y, X should be a super key of relation R.
Dependency	3NF can be obtained without sacrificing all dependencies.	Dependencies may not be preserved in BCNF.
Decomposition	Lossless decomposition can be achieved in 3NF.	Lossless decomposition is hard to achieve in BCNF.

e- What is 2 face locking(2PL)? Describe with the help of example?

Ans-A transaction is said to follow Two Phase Locking protocol if Locking and Unlocking can be done in two phases.

1. **Growing Phase:** New locks on data items may be acquired but none can be released.
2. **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.

Note – If lock conversion is allowed, then upgrading of lock(from S(a) to X(a)) is allowed in Growing Phase and downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.
Let’s see a transaction implementing 2-PL.

	T_1	T_2
1	LOCK-S(A)	
2		LOCK-S(A)
3	LOCK-X(B)	

4	**…….**	**……**
5	**UNLOCK(A)**	
6		**LOCK-X(C)**
7	**UNLOCK(B)**	
8		**UNLOCK(A)**
9		**UNLOCK(C)**
10	**…….**	**……**

This is just a skeleton transaction which shows how unlocking and locking works with 2-PL. Note for:

Transaction T_1:

- Growing Phase is from steps 1-3.
- Shrinking Phase is from steps 5-7.
- Lock Point at 3

Transaction T_2:

- Growing Phase is from steps 2-6.
- Shrinking Phase is from steps 8-9.
- Lock Point at 6

2017-18

SECTION-C

3. a)

SERIALIZABILITY

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

CONFLICT SERIALIZABILITY

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

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

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

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

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

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

<u>schedule 3:</u>

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

here the write(A) instruction of T_1 conflicts with read(A) instructions of T_2, However, the write(A) of T_2 does not conflict with the read(B) instruction of T_1.

If we continue the swaping of nonconflicting instructions:

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

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

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

The final result of these swaps is <u>schedule 6</u> a serial schedule. Thus schedule 3 is equivalent to a serial schedule. If a schedule S can be transformed into a schedule S' by a series of swaps of nonconflicting instructions, we say that S and S' are conflict equivalent.

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

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

VIEW SERIALIZABILITY

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

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

S_1

T_1	T_2
r(A)	
	r(B)

S_2

T_1	T_2
	r(B)
r(A)	

$S_1 = S_2$

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

S_1

T_1	T_2	T_3
	W(A)	
r(A)		
		W(A)

S_2

T_1	T_2	T_3
		W(A)
	W(A)	
r(A)		

$S_1 = S_2$

3. Final Write should be same in both schedule.
For each data item Q, the transaction that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'.

S_1

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

S_2

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

$S_1 \neq S_2$

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

for Example:-

Schedule S

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

Schedule U

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

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

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

Testing for Serializability of a schedule.

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

Algorithm:- Testing conflict serializability of a Schedule S.

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

4. For each case in S where T_j executes a write(X) after T_i executes a write(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

3. b)

MULTIVERSION SCHEMES:

In multiversion concurrency-control schemes, each write(Q) operation creates a new version of Q. When a transaction issues a read(Q) operation, the concurrency control manager selects one of the versions of Q to be read. The concurrency-control scheme must ensure that the version to be read is selected in a manner that ensures serializability. It is also crucial, for performance reasons, that a transaction be able to determine easily and quickly which version of the data item should be read.

Multiversion Timestamp Ordering

The timestamp-ordering protocol can be extended to a multiversion protocol. With each transaction T_i in the system, we associate a unique static timestamp, denoted by $TS(T_i)$. The database system assigns this timestamp before the transaction starts execution.

With each data item Q, a sequence of versions $\langle Q_1, Q_2, \ldots, Q_m \rangle$ is associated. Each version Q_k contains three data fields:

- Content is the value of version Q_K.
- W-timestamp(Q_K) is the timestamp of the transaction that created version Q_K.
- R-timestamp(Q_K) is the largest timestamp of any transaction that successfully read version Q_K.

A transaction - say, T_i - creates a new version Q_K of data item Q by issuing a write(Q) operation. The content field of the version holds the value written by T_i. The system initializes the W-timestamp and R-timestamp to $TS(T_i)$. It updates the R-timestamp value of Q_K whenever a transaction T_j reads the content of Q_K, and R-timestamp $(Q_K) < TS(T_j)$.

The multiversion timestamp-ordering scheme presented next ensures serializability. The scheme operates as follows:

Suppose that Transaction T_i issues a read(Q) or write(Q) operation. Let Q_K denote the version of Q whose write timestamp is the largest write timestamp less than or equal to $TS(T_i)$. W-timestamp $(Q_K) <= TS(T_i)$.

1. If Transaction T_i issues a read(Q), then the value returned is the content of version Q_K.

2. If transaction T_i issues write(Q), and if $TS(T_i) <$ R-timestamp(Q_k), then the system rolls back transaction T_i.
On the other hand, if $TS(T_i) =$ W-timestamp(Q_k), the system overwrites the contents of Q_k; otherwise (if $TS(T_i) >$ R-timestamp(Q_k)), it creates a new version of Q.

The justification for rule 1 is clear. A transaction reads the most recent version that comes before it in time.
The second rule forces a transaction to abort if it is "too late" in doing a write. More precisely, if T_i attempts to write a version that some other transaction would have read, then we cannot allow that write to succeed.

TIMESTAMP-BASED PROTOCOL

A method for determining the serializability order is to select an ordering among transactions in advance. The most common method for doing so is to use a timestamp-ordering scheme.

THE TIMESTAMP-ORDERING PROTOCOL :-

The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows:

① Suppose that transaction T_i issue read(Q)

ⓐ If $TS(T_i) <$ W-timestamp(Q), then T_i needs to read a value of Q that was already overwritten. Hence read operation is rejected, and T_i is rolled back.

ⓑ If $TS(T_i) \geq$ W-timestamp(Q), then the read operation is executed, and R-timestamp(Q) is set to maximum of R-timestamp(Q) and $TS(T_i)$.

② Suppose that T_i issue write(Q)

ⓐ If $TS(T_i) <$ R-timestamp(Q), then the value of Q that T_i is producing was needed previously, and the system assumed that value would never be produced. Hence the system rejects the write operation and rolls T_i back

ⓑ If $TS(T_i) <$ W-timestamp(Q), then T_i is attempting to write an obsolete value Q. Hence the system rejects this write operation and rolls T_i back.

ⓒ otherwise, the system executes the write operation and sets W-timestamp(Q) to $TS(T_i)$.

* The timestamp-ordering protocol ensures conflict serializability.
* It ensures freedom from deadlock.
* there is possibility of starvation.

4. a)

i) select RollNo, Name from Student where Branch = 'CSE';

ii) select Name from Student INNER JOIN Issue ON Student.RollNo= Issue.RollNo INNER JOIN Book ON Issue.Isbn = Book.Isbn Where Book.publisher = 'BPB';

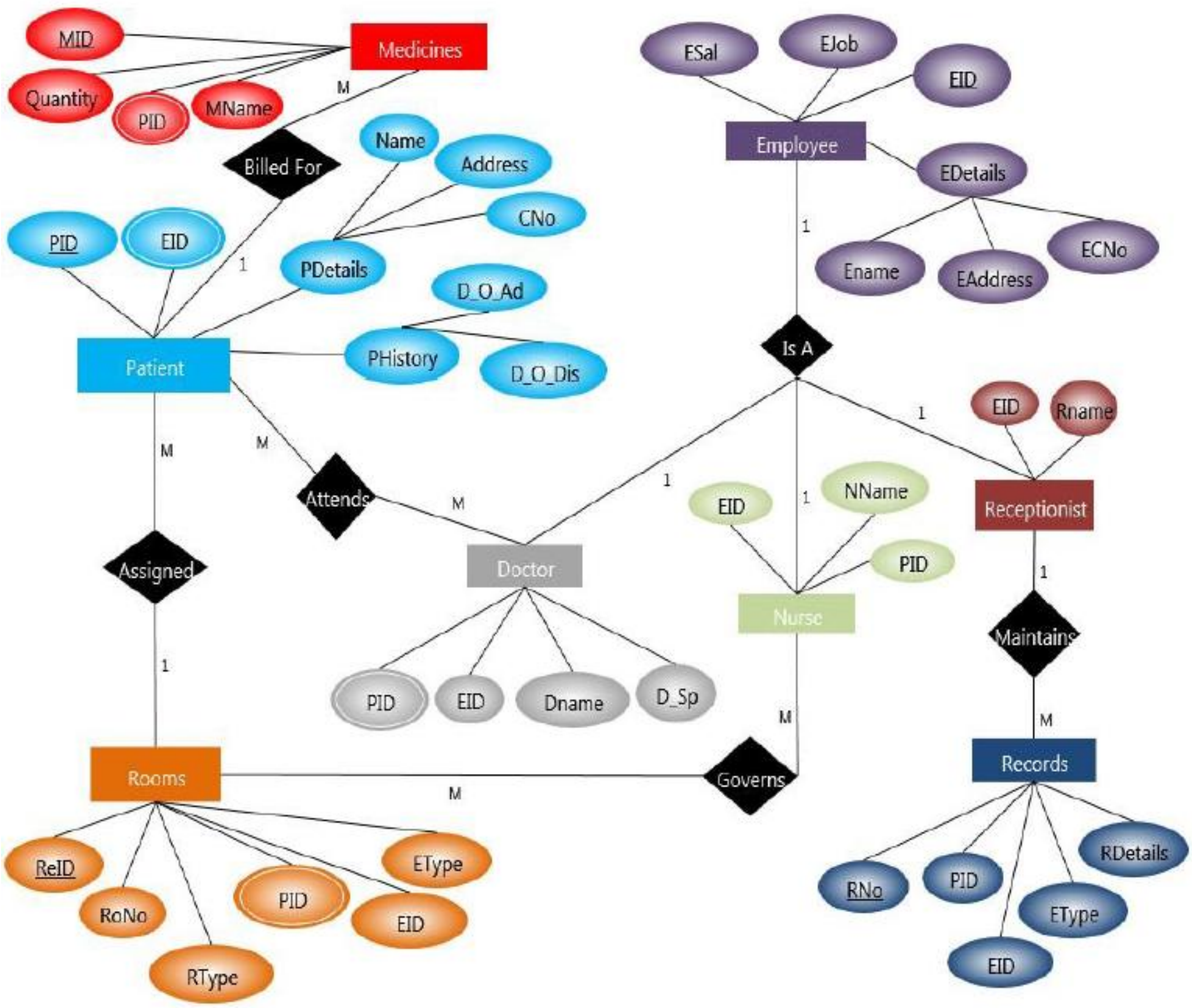
iii) select title,author from Student INNER JOIN Issue ON Student.RollNo= Issue.RollNo INNER JOIN Book ON Issue.Isbn = Book.Isbn Where Student.name LIKE 'a%';

iv) select title from Book INNER JOIN Issue ON Book.Isbn = Issue.Isbn Where te_of_issue <= 20/09/2012;

v) select name from Student INNER JOIN Issue ON Student.RollNo= Issue.RollNo INNER JOIN Book ON Issue.Isbn = Book.Isbn Where Author = 'Sanjeev';

4. b)

ER Diagram for Hospital Management System is shown below:



Tables

Table 1 – <u>employee</u>

Attribute	Description	Data type	Condition
EID	Employee ID	Varchar2	Primary Key
EName	Employee Name	Varchar2	
EAddress	Employee Address	Varchar2	
ECNo	Contact Number	Number	
EJob	Job Description	Varchar2	
Esal	Employee Salary	Number	

Table 2 – patient

Attribute	Description	Data type	Condition
PID	Patient ID	Varchar2	Primary Key
EID	Employee ID	Varchar2	Foreign Key (ref – employee)
Name	Patient Name	Varchar2	
Address	Patient Address	Varchar2	
CNo	Contact Number	Number	
D_O_Ad	Date Of Admission	Varchar2	
D_O_Dis	Date Of Discharge (Probable)	Varchar2	

Table 3 – doctor

Attribute	Description	Data type	Condition
EID	Employee ID	Varchar2	Foreign Key (ref – employee)
PID	Attending Patient(s) ID	Varchar2	Foreign Key (ref – patient)
DName	Doctor's Name	Varchar2	
D_Sp	Specialization	Varchar2	

Table 4 – nurse

Attribute	Description	Data type	Condition
EID	Employee ID	Varchar2	Foreign Key (ref – employee)
PID	Patient ID	Varchar2	Foreign Key (ref – patient)
NName	Nurse's Name	Varchar2	

Table 5 – receptionist

Attribute	Description	Data type	Condition
EID	Employee ID	Varchar2	Foreign Key (ref – employee)
RName	Receptionist's Name	Varchar2	

Table 6 – records

Attribute	Description	Data type	Condition
RNo	Record Number	Varchar2	Primary Key
PID	Patient ID	Varchar2	Foreign Key (ref – patient)
EID	Employee ID	Varchar2	Foreign Key (ref – employee)
EType	Employee Type	Varchar2	
RDetails	Record Details	Varchar2	

Table 7 – rooms

Attribute	Description	Data type	Condition
ReID	Room Record ID	Varchar2	Primary Key
RoNo	Room Number	Number	
Rtype	Room Type	Varchar2	
PID	Patient ID	Varchar2	Foreign Key (ref – patient)
EID	Employee ID	Varchar2	Foreign Key (ref – employee)
EType	Employee Type	Varchar2	

Table 8 – medicines

Attribute	Description	Data type	Condition
MID	Medicine ID	Varchar2	Primary Key
PID	Patient ID	Varchar2	Foreign Key (ref – patient)
Quantity	Quantity	Number	
MName	Medicine Name	Varchar2	

SQL DDL Commands

1. Creating Table **employee** –

```
SQL> create table employee (eid varchar2(5) primary key,ename varchar2(10),eaddr
ess varchar2(10),ecno number(10),ejob varchar2(10),esal number(6));

Table created.
```

2. Creating Table **patient** –

```
SQL> create table patient (pid varchar2(5) primary key,eid varchar2(5) reference
s employee(eid),name varchar2(10),address varchar2(10),cno number(10),d_o_ad dat
e,d_o_dis date);

Table created.
```

3. Creating Table **doctor** –

```
SQL> create table doctor (eid varchar2(5) references employee(eid),pid varchar2(
5) references patient(pid),dname varchar2(10),d_sp varchar2(10));

Table created.
```

4. Creating Table **nurse** –

```
SQL> create table nurse (eid varchar2(5) references employee(eid),pid varchar2(5
) references patient(pid),nname varchar2(10));

Table created.
```

5. <u>Creating Table **rooms**</u> –

```
SQL> create table rooms (ReID varchar2(5) primary key,RoNo number(3),RType varch
ar2(8),PID varchar2(5) references patient(pid),eid varchar2(5) references hemplo
yee(eid),etype varchar2(10));

Table created.
```

6. <u>Creating Table **receptionist**</u> –

```
SQL> create table receptionist (eid varchar2(5) references employee(eid),rname v
archar2(10));

Table created.
```

7. <u>Creating Table **records**</u> –

```
SQL> create table records (RNo varchar2(5) primary key,PID varchar2(5) reference
s patient(pid),eid varchar2(5) references hemployee(eid),etype varchar2(10),Rdet
ails varchar2(10));

Table created.
```

8. <u>Creating Table **medicines**</u> –

```
SQL> create table medicines (mid varchar2(5) primary key,pid varchar2(5) referen
ces patient(pid),quantity number(4),mname varchar2(10));

Table created.
```

5a) Explain the Primary Key, Super Key, Foreign Key and Candidate key with example.

KEYS

(iii) Primary Key :- It denotes a key that is chosen by the database designer as the principal means of identifying unique records within a table. The primary key should be chosen in such a way that its values must not (or rarely) change. For example, the employee's code field can be designated as the primary key because all employee code are unique & the value once entered is ~~not~~ never changed.

records in ascending or descending order.

(v) Foreign or Reference key :- A foreign key is an attribute or combination of attribute in one base table that points to the primary key of another table. The purpose of the foreign key is to ensure referential integrity of the data i.e only values that are supposed to appear in the database are permitted.

Ex:- Consider two tables with fields i.e

Department (dno (PK), dname) & Employee (id, name, address, salary, dno)

In Employee table dno is Foreign key that points to the Primary key (dno) of department table.

— [illegible] in above example, {id}, {name, address} are Candidate keys, since key are sufficient to ~~[illegible]~~ distinguish the values.

5b) Short Notes of the Following- i) MVD or JD

Multivalued dependency

Multivalued dependencies occurs when two or more independent multi-valued facts about the same attribute occur within the same relation. It is denoted by $\alpha \twoheadrightarrow \beta$.

Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency $\alpha \twoheadrightarrow \beta$ holds on R if, in any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exists tuples t_3 and t_4 in r such that

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R-\beta] = t_2[R-\beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R-\beta] = t_1[R-\beta]$$

⇒ If multivalued dependency $\alpha \twoheadrightarrow \beta$ is satisfied by all relations on schema R, then $\alpha \twoheadrightarrow \beta$ is a trivial multivalued dependency on schema R. Thus, $\alpha \twoheadrightarrow \beta$ is trivial if $\beta \subseteq \alpha$ or $\beta \cup \alpha = R$.

e.g. R= (<u>loan_no, customer_id</u>, customer_street, customer_city)

loan_no	customer_id	customer_street	customer_city
L-23	99-123 t_1	North	Rye
L-23	99-123 t_2	Main	Manchester
L-27	99-123 t_3	North	Rye
L-27	99-123 t_4	Main	Manchester
L-93	15-106	Lake	Horseneck

Join Dependency

Let R be a relation schema and $R_1, R_2, \ldots, R_n$ be the decomposition of R; R is said to satisfy the join dependency $*(R_1, R_2, \ldots, R_n)$ (read as "star $R_1, R_2, \ldots, R_n$"), if and only if

$$\pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \cdots \bowtie \pi_{R_n}(r) = r$$

In other words, relation schema R satisfied the JD $*(R_1, R_2, \ldots, R_n)$ if and only if every legal relation $r(R)$ is equal to join of its projections on $R_1, R_2, \ldots R_n$.

A join dependency $*(R_1, R_2, \ldots, R_n)$ specified on a relation schema R is a trivial JD, if at least one relati schema R_i in JD $*(R_1, R_2, \ldots, R_n)$ is the set of all attribu of R (that is, one of the relation schemas R_i in JD $*(R_1, R_2, \ldots, R_n)$ is equal to R). Such a dependency is called trivial ~~because~~ JD.

5b)ii)Normalization advantages

Database normalization is a process in which we modify the complex database into the simpler database.

Advantages of normalization?

1. **Data consistency**

- Data consistency means that the data is always real and it is not ambiguous.

Data becomes nonredundant

- Non-redundant means that only copy original copy of data is available for each user and for every time. There are no multiple copies of the same data for different persons. So when data is changed in one file and stay in one file. Then of course data is consistent and non-redundant. Here redundant is not the same as a backup of data, both are different things.

Reduce insertion, deletion and updating anomalies

- **Insertion anomaly** is an anomaly that occurs when we want to insert data into the database but the data is not completely or correctly inserted in the target attributes. If completely inserted in the database then not correctly entered.
- **Deletion anomaly** is an anomaly that occurs when we want to delete data in the database but the data is not completely or correctly deleted in the target attributes.
- **Updation anomaly** is an anomaly that occurs when we want to update data in the database but the data is not completely or correctly updated in the target attributes.

Database table compaction

- When we normalize the database, we convert the large table into a smaller table that leads to data and table compaction. Compaction means to have the least and required size.

Better performance

Fast queries

6a) What is Log? How is it maintained? Discuss the features of deferred database modification and immediate database modification in brief.

LOG-BASED RECOVERY

The most widely used structure for recording database modifications is the log. The log is a sequence of log records; recording all the update activities in the database. There are several types of log records. An update log record describes a single database write. It has these fields:

Transaction Identifier is the unique identifier of the transaction that performed the write operation.

Data-item identifier is the unique identifier of the data item written. Typically it is the location on disk of the data item.

Old value is the value of the data item prior to the write.

New value is the value that the data item will have after the write.

e denote the various types of log records as:

<T_i start>. Transaction T_i has started.

<T_i, X_j, V_1, V_2>. Transaction T_i has performed a write on data item X_j. X_j had value V_1 before the write, and will have value V_2 after the write.

<T_i commit>. Transaction T_i has commited.

<T_i abort>. Transaction T_i has aborted.

<u>Immediate Database Modification</u>

The immediate-modification technique allows database modifications to be output to the database while the transaction is still in the active state. Data modifications written by active transactions are called <u>uncommitted modifications</u>. In the event of a crash or a transaction failure, the system must use the old-value field of the log to restore the modified data item to the value they had prior to the start of the transaction

Now, portion of the system log corresponding to T_0 and T_1

$\langle T_0 \text{ start} \rangle$
$\langle T_0, A, 1000, 950 \rangle$
$\langle T_0, B, 2000, 2050 \rangle$
$\langle T_0 \text{ commit} \rangle$
$\langle T_1 \text{ start} \rangle$
$\langle T_1, C, 700, 600 \rangle$
$\langle T_1 \text{ commit} \rangle$

Now,

Log	Database
$\langle T_0 \text{ start} \rangle$	
$\langle T_0, A, 1000, 950 \rangle$	
$\langle T_0, A, 2000, 2050 \rangle$	
	$A = 950$
	$B = 2050$
$\langle T_0 \text{ commit} \rangle$	
$\langle T_1 \text{ start} \rangle$	
$\langle T_1, C, 700, 600 \rangle$	$C = 600$
$\langle T_1 \text{ commit} \rangle$	

Using the log the system can handle any failure that does not result in the loss of information in non-volatile storage. The recovery scheme uses two recovery procedure:

- undo(T_i) restores the value of all data items updated by transaction T_i to the old values.
- redo(T_i) sets the value of all data items updated by transaction T_i to the new values.

After a failure has occured, the recovery scheme consults the log to determine which transactions need to be redone, and which need to be undone.

- Transaction T_i needs to be <u>undone</u> if the log contains the record $\langle T_i \text{ start} \rangle$ but does not contain $\langle T_i \text{ commit} \rangle$
- Transaction T_i needs to be <u>redone</u> if the log contains both the record $\langle T_i \text{ start} \rangle$ and $\langle T_i \text{ commit} \rangle$

e.g.

$\langle T_0 \text{ start} \rangle$	$\langle T_0 \text{ start} \rangle$	$\langle T_0 \text{ start} \rangle$
$\langle T_0, A, 1000, 950 \rangle$	$\langle T_0, A, 1000, 950 \rangle$	$\langle T_0, A, 1000, 950 \rangle$
$\langle T_0, B, 2000, 2050 \rangle$	$\langle T_0, B, 2000, 2050 \rangle$	$\langle T_0, B, 2000, 2050 \rangle$
↓	$\langle T_0 \text{ commit} \rangle$	$\langle T_0 \text{ commit} \rangle$
undo(T_0)	$\langle T_1 \text{ start} \rangle$	$\langle T_1 \text{ start} \rangle$
	$\langle T_1, C, 700, 600 \rangle$	$\langle T_1, C, 700, 600 \rangle$
	↓	$\langle T_1 \text{ commit} \rangle$
	redo(T_0)	↓
	undo(T_1)	redo(T_0)
		redo(T_1)

whenever a transanction performs a write, it is essential that the log record for that write be created before the database is modified. Once a log record exists, we can output the modification to the database if that is desirable. Also we have the ability to undo a modification that has already been output to the database. we undo it by using the old-value field in log record.

Deffered database modification

The differed database-modification technique ensures transaction atomicity by recording all database modification in the log, but differing the execution of all write operations of a transaction until the transaction partially commits. In this technique we assume that transactions are executed serially.

when a transaction partially commits, the information n the log associated with the transaction is used in executing the diferred writes. If the system crashes before the transaction completes its execution, or if the transaction aborts, then the information on the log is simply ignored.

The execution of transaction Ti proceeds as follows. Before Ti starts its execution, a record <Ti, start> is written to the log. A write(x) operation by Ti results in the writing a new record to the log. Finally when Ti ~~partially~~ commits, a record <Ti commit> is written to the log.

When transaction partially commits, the records associated with it in the log are used in executing deferred ~~log~~ writes. Since a failure may occur while this updating is taking place, we must ensure that, before the start of these updates, all the log records are written out to stable storage.

Now consider two transactions T_0 and T_1

T_0: read(A);	T_1: read(C);
A:= A-50;	C:= C-100;
write(A);	write(C);
read(B);	
B:= B+50;	
write(B);	

suppose that these transactions are executed serially, in the order T_0 followed by T_1; and values of A, B, and C before executions took place were 1000, 2000 & 700

Now portion of database log corresponding d . and

<T_0 start>
<T_0, A, 950>
<T_0, B, 2050>.
<T_0 commit>
<T_1 start>
<T_1, C, 600>
<T_1 commit>

State of the log and database corresponding to T_0 and T_1

log	database
<T_0 start>	
<T_0, A, 950>	
<T_0, B, 2050>	
<T_0 commit>	
	A = 950
	B = 2050
<T_1 start>	
<T_1, C, 600>	
<T_1 commit>	
	C = 600

Using the log, the system can handle any failure that results in the loss of information on volatile storage. The recovery scheme uses the following recovery procedure:

- redo(T_i) sets the value of all data items updated by transaction T_i to the new values.

The redo operation must be idempotent; that is, executing it several times must be equivalent to executing it once.

After a failure, the recovery subsystem consults the log to determine which transactions need to be redone. Transaction T_i need to be redone if and only if the log contains both the record $\langle T_i \text{ start} \rangle$ and $\langle T_i \text{ commit} \rangle$

e.g.

$\langle T_0 \text{ start} \rangle$	$\langle T_0 \text{ start} \rangle$	$\langle T_0 \text{ start} \rangle$
$\langle T_0, A, 950 \rangle$	$\langle T_0, A, 950 \rangle$	$\langle T_0, A, 950 \rangle$
$\langle T_0, B, 2050 \rangle$	$\langle T_0, B, 2050 \rangle$	$\langle T_0, B, 2050 \rangle$
	$\langle T_0 \text{ commit} \rangle$	$\langle T_0, \text{commit} \rangle$
$\downarrow$	$\langle T_1 \text{ start} \rangle$	$\langle T_1 \text{ start} \rangle$
no redo takes place	$\langle T_1, C, 600 \rangle$	$\langle T_1, C, 600 \rangle$
	$\downarrow$	$\langle T_1 \text{ commit} \rangle$
	redo (T_0)	$\downarrow$
		redo(T_0) ✓
		& redo(T_1) ✓

b) What do you mean by Transaction? Explain transaction property with detail and suitable example.

TRANSACTION SYSTEM

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

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

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

TRANSACTION CONCEPT

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

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

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

These operations are often called the "ACID" properties.

Transactions access data using two operations:

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

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

$$
\begin{aligned}
T_i:\ & read(A);\\
& A := A - 50;\\
& write(A);\\
& read(B);\\
& B := B + 50;\\
& write(B).
\end{aligned}
$$

ATOMICITY

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

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

CONSISTENCY

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

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

ISOLATION

Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions T_i and T_j, it appears to T_i that either T_j finished execution before T_i started or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

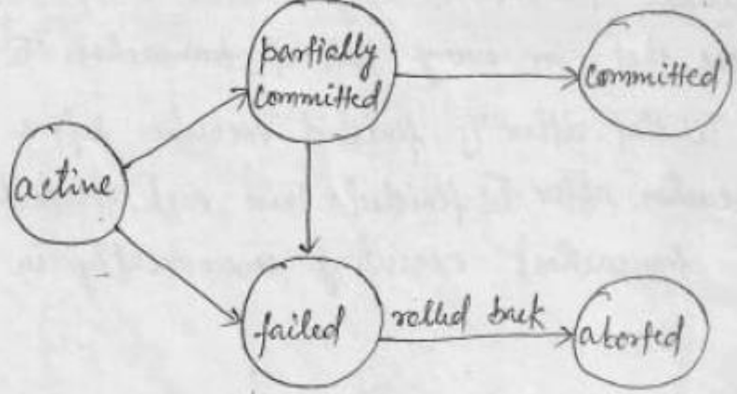
DURABILITY

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

TRANSACTION STATE

A transaction must be in one of the following states:

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



STATE DIAGRAM OF A TRANSACTION

A transaction enters the failed state after the system determines that the transaction can no longer proceed with its normal execution. Such a transaction must be rolled back. Then, it enters the aborted state. At this point, the system has two options:

* It can restart the ~~system~~ transaction, but only if the transaction was aborted as a result of some h/w or s/w error.
* It can kill the transaction if the transaction was aborted due to some logical error.

7a Explain all database languages in detail with example.

DDL - set of SQL commands used to create, modify, and delete database structure but not data. They are normally used by the DBA

Commands -

① creating database object

create table <table name> (fieldname, Datatype, (width), ----);

create table student (name varchar2 (12), Age number);

creating table using constraint

Age number NOTNULL.

② modify the database

Alter table employ add (salary (12));

Alter table employ Drop column salary;

③ Droping an object (table)

Drop table employ;

DML → Procedural DML → specify what data are needed how to get those data (select) command

DML → non procedural DML → what data are needed.

commands - insert update, delete and query the data.

insert into table values ("Ak", 12); 12

select * from employ; (retrieve data)

update employ set age = '16' where name = 'Ak';
updating the rows.

DCL — commit, Rollback command.

7b) Explain data fragmentation with types.

2) Data Fragmentation -

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

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

$S_1, S_2, S_3 - - - S_n$

There are three types of fragmentation:

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

soln - we have three types of fragmentation

1) Horizontal Fragmentation -

tuples of relation are divided into number of subsets.

Employ table

ex-

Emp no	Emp_name	Designation	Salary	Dept_no
1001	Amit	CA	15000	1
1002	Anu	CEO	25000	1
1003	Ravi	MD	20000	2
1004	Rahul	PRO	10000	2

for horizontal fragmentation

$Employ1 = \sigma_{Dept\_no = "1"}(Employ)$

$Employ2 = \sigma_{Dept\_no = "2"}(Employ)$

Thus we have two fragments

Relation Employ1

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

2) Relation Employ 2

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

resonstruction we got by onion of relations
Employ = Employ1 ∪ Employ2

Vertical Fragmentation –

attributes are divided into number of subsets.

ex-

Emp

Emp no	Emp_name	Salary	Dept_no	Tuple_id
1001	Amit	15000	1	1
1002	Anu	25000	1	2
1003	Ravi	20000	2	3
1004	Rahul	10000	2	4

here we add tupe-id which shows logical address of every tuple.
for ex - we want to fragments of Emp these are $Empv_1$ $EmpV_2$ (vertically)

$Empv_1 = \Pi_{Emp\_no,\ Emp\_name,\ Tuple\_id} (Emp)$

$Empv_2 = \Pi_{Salary,\ Deptno,\ Tuple\_id} (Emp)$

these relation shown as

Relation $Empv_1$

Emp_no	Emp_name	Tuple_id
1001	Amit	1
1002	Anu	2
1003	Ravi	3
1004	Rahul	4

Relation $EmpV_2$

Salary	Dept_no	Tuple_id
15000	1	1
25000	1	2
20000	2	3
10000	2	4

To reconstruct Emp we join $Empv_1$ and $Empv_2$

i.e $EmpV_1 \times EmpV_2$

Mixed Fragmentation horizontal of relation followed by vertical fragmentation.

$$\Pi_{Emp\_name, salary}(\sigma_{emp\_no > 1003}(Emp))$$

result is

Emp_name	Salary
Rahul	10000