1. List any three characteristics of database system

1.Self-Describing Nature of a Database System

Database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog Information stored in the catalog is called metadata and it describes the structure of the primary database

2.Insulation between Programs and Data, and Data Abstraction

The structure of data files is stored in the DBMS catalog separately from the access programs.This property is called program-data independence .An operation (also called a function or method) is specified in two parts.- Interface and Implementation. The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters). The implementation (or method) of the operation is specified separately and can be changed without affecting the.User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence. The characteristic that allows program-data independence and program operation independence is called data abstraction.

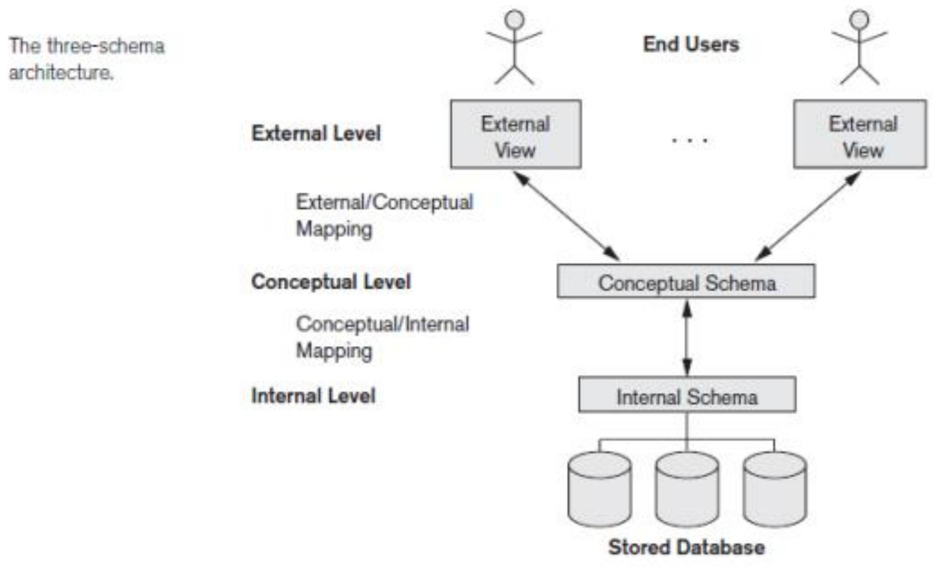
3.Support of Multiple Views of the Data

A database has many users, each user may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.

4.Sharing of Data and Multiuser Transaction Processing

DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct

2. Draw neat labelled diagram of three schema architecture and briefly describe each level



Internal level

The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

Conceptual level

The conceptual schema describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. The implementation conceptual schema is often based on a conceptual schema design in a high-level data model

External or view level

The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. Each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

3. Write briefly about any three relational database integrity constraints.

Key constraints

Superkey of R: A super key is a group of single or multiple keys which identifies rows in a table.

Key of R: It is a "minimal" superkey. That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey Relation may have more than one key. In such cases each of the keys are called candidate keys If a relation has several candidate keys, one is chosen arbitrarily to be the primary key.

Entity integrity constraints

The primary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R).

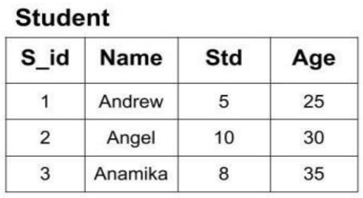
Referential integrity constraints

This constraint involves two relations . Used to specify a relationship among tuples in two relations: -1. Referencing relation and 2. Referenced relation. Tuples in the referencing relation R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R2. A tuple t1 in R1 is said to reference a tuple t2 in R2 if t1[FK] = t2[PK].

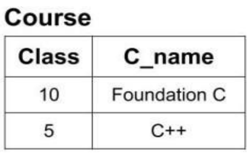
4. Differentiate between theta join and natural join operations.

Theta join is a join which combines the tuples from different relations according to the given theta condition.The join condition in theta join is denoted by theta(θ) symbol. θ (theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$.

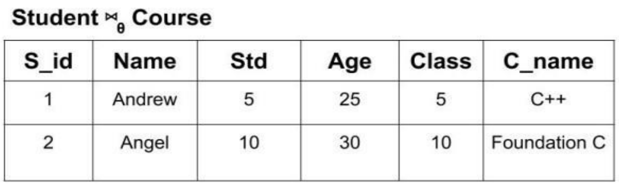
Notation:$R1 \bowtie_{\theta} R2$,where R1 and R2 are relations such that they don't have any common attribute.


Student

S_id	Name	Std	Age
1	Andrew	5	25
2	Angel	10	30
3	Anamika	8	35


Course

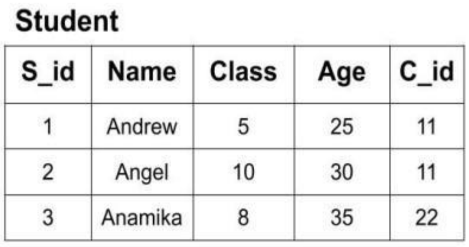
Class	C_name
10	Foundation C
5	C++


Student $\bowtie_\theta$ Course

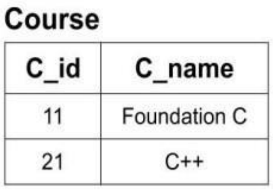
S_id	Name	Std	Age	Class	C_name
1	Andrew	5	25	5	C++
2	Angel	10	30	10	Foundation C

Natural join requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, rename operation has to be applied first to convert the name of the join attribute of one relation same as that of the join attribute in second relation.

Notation: R1 R2* where R1 and R2 are two relations

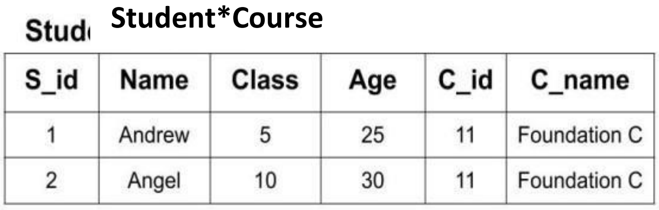

Student

S_id	Name	Class	Age	C_id
1	Andrew	5	25	11
2	Angel	10	30	11
3	Anamika	8	35	22


Course

C_id	C_name
11	Foundation C
21	C++

Student*Course



S_id	Name	Class	Age	C_id	C_name
1	Andrew	5	25	11	Foundation C
2	Angel	10	30	11	Foundation C

5. Give any three uses of a trigger

- Enforce business rules
- Validate input data
- Generate a unique value for a newly inserted row on a different file
- Write to other files for audit trail purposes
- Query from other files for cross-referencing purposes
- Access system functions (for example, print an exception message when a rule is violated)

- Replicate data to different files to achieve data consistency

6. A file has r =20000 STUDENT records of fixed length. Each record has the following fields: NAME (30 bytes), SSN (9 bytes), ADDRESS (40 bytes), PHONE(9 bytes), BIRTHDATE (8 bytes), GENDER (1 byte), DEPTID (4 bytes), CLASSCODE (4 bytes), and PROGID (3 bytes). An additional byte is used as a deletion marker. The file is stored on the disk with block size B=512 bytes,
Calculate the record size R in bytes.Calculate the blocking factor bfr and the number of file blocks b assuming an unspanned organization.Calculate the average time it takes to find a record by doing a linear search

No. of records r =20000

Block size B=512 bytes

Record size R=30+9+40+9+8+1+4+4+3+1=109 bytes

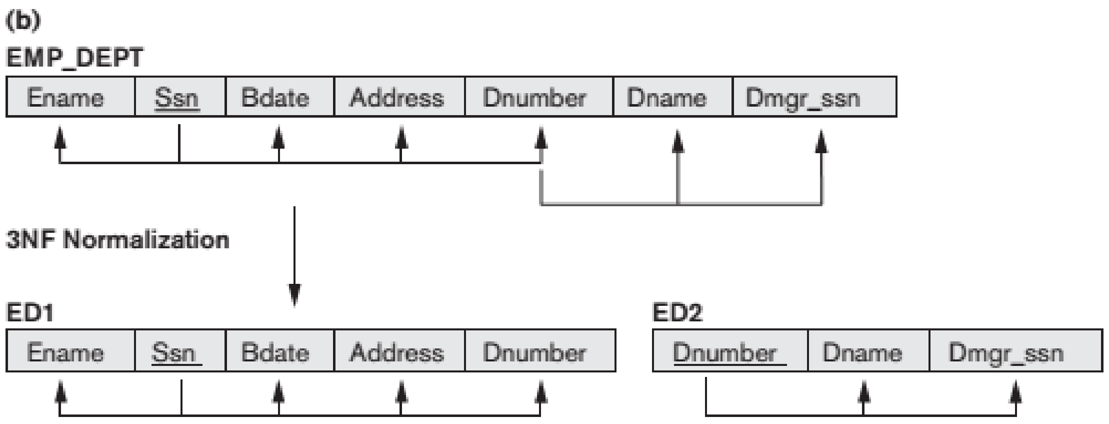
Blocking factor bfr=B/R=512/109 =4.69=4

Number of file blocks b=r/bfr=20000/4=5000

Average time to find a record by doing a linear search=b/2=2500

7. Define Boyce-Codd normal form. How does it differ from 3NF?

A relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.



BCNF is an advance version of 3NF that's why it is also referred as 3.5NF.BCNF is stricter than 3NF.A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF

8. Suppose, a relational schema R (P,Q, R, S) and set of functional dependencies F and G are as follow:F : { P → Q, Q → R, R → S } G : { P → QR, R → S }. Check the equivalency of functional dependencies F and G

Checking if F can be inferred from G:

To infer F from G, we need to check if each dependency in F can be derived using the dependencies in G.

a) P → Q:

In G, we have P → QR. To derive P → Q, we can apply the augmentation rule and remove R from the right-hand side, giving us P → Q. Thus, this dependency can be inferred from G.

b) Q → R:

In G, we have P → QR, but we don't have a direct dependency from Q to R. Therefore, we cannot infer Q → R from G.

c) R → S:

In G, we have R → S. Thus, this dependency can be directly inferred from G.

Since Q → R cannot be derived from G, F cannot be completely inferred from G.

Checking if G can be inferred from F:

To infer G from F, we need to check if each dependency in G can be derived using the dependencies in F.

a) P → QR:

In F, we have P → Q and Q → R. By transitivity, we can combine these two dependencies to infer P → QR. Therefore, this dependency can be derived from F.

b) R → S:

In F, we have R → S. Thus, this dependency can be directly inferred from F.

Since all the dependencies in G can be derived from F, G can be completely inferred from F.

Based on the analysis, we can conclude that G can be inferred from F, but F cannot be completely inferred from G. Therefore, the functional dependencies F and G are not equivalent.

9. Write briefly on log based recovery

Recovery from transaction failures usually means that the database is restored to the most recent consistent state just before the time of failure.To do this, the system must keep information about the changes that were applied to data items by the various transactions.This information is typically kept in the system log.

Log is a sequence of records, which maintains the records of actions performed by a transaction.It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is fai lsafe.

Log –based recovery works as follows:

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it as <Tn, Start>.
- When the transaction modifies an item X, it write logs as follows:

  <Tn,X,V1,V2 >

It reads Tn has changed the value of X, from V1 to V2

- When the transaction finishes, it logs : <Tn, commit>

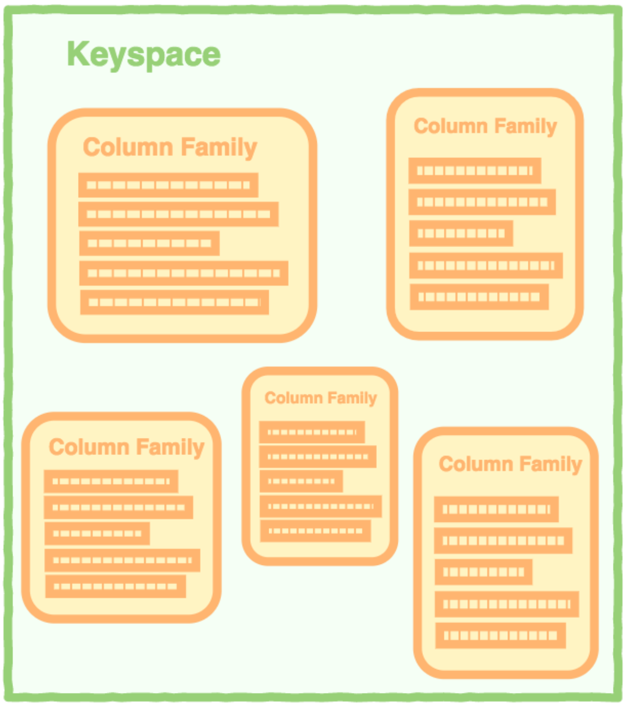
After a system crash has occurred, the system consults the log to determine which transactions need to be redone, and which need to be undone so as to ensure atomicity.

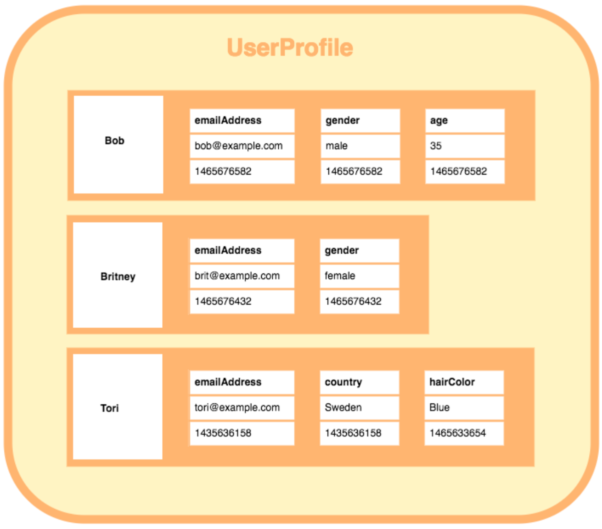
- Transaction Ti needs to be undone if the log contains the record <Ti,start>, but does not contain either the record <Ti, commit> or the record <Ti,abort>.
- Transaction Ti needs to be redone if the log contains the record <Ti,start> and either the record <Ti, commit> or the record <Ti,abort>.

10. Explain briefly the characteristics of Column family database

Columns store databases use a concept called a *keyspace*. A keyspace is kind of like a schema in the relational model. The keyspace contains all the column families (kind of like tables in the relational model), which contain rows, which contain columns.

Like this:



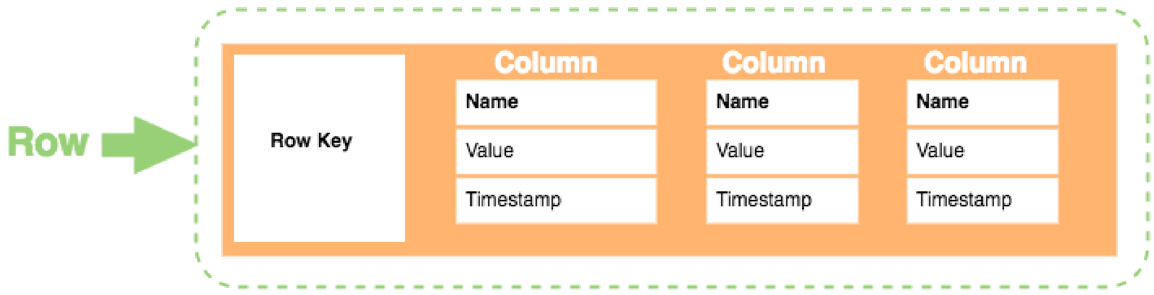


As the above diagram shows:

- A **column family** consists of multiple rows.

- Each **row** can contain a different number of columns to the other rows. And the columns don't have to match the columns in the other rows (i.e. they can have different column names, data types, etc).
- Each **column** is contained to its row. It doesn't span all rows like in a relational database. Each column contains a name/value pair, along with a timestamp. Note that this example uses Unix/Epoch time for the timestamp.

Here's how each row is constructed:



Here's a breakdown of each element in the row:

- **Row Key**. Each row has a unique key, which is a unique identifier for that row.
- **Column**. Each column contains a name, a value, and timestamp.
- **Name**. This is the name of the name/value pair.
- **Value**. This is the value of the name/value pair.
- **Timestamp**. This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

Advantages of column-oriented databases

- **Scalability.** This is a major advantage and one of the main reasons this type of database is used to store big data. With the ability to be spread over hundreds of different machines depending on the scale of the database, it supports massively parallel processing. This means it can employ many processors to work on the same set of computations simultaneously.

- **Compression.** Not only are they infinitely scalable, but they are also good at compressing data and thus saving storage.
- **Very responsive.** The load time is minimal, and queries are performed fast, which is expected given that they are designed to hold big data and be practical for analytics.

11 a) Differentiate between two-tier and three-tier client-server database architecture with the help of neat labelled diagrams.

Two-Tier Client/Server Architectures for DBMS

Server handles Query and transaction functionality related to SQL processing. Server is often called query server or transaction server because it provides these two functionalities. Client handles User interface programs and application programs. For DBMS access, program establishes a connection to the server. Open Database Connectivity (ODBC) provides application programming interface (API) Allows client-side programs to call the DBMS. Both client and server machines must have the necessary software installed. JDBC allows Java client programs to access one or more DBMSs through a standard interface

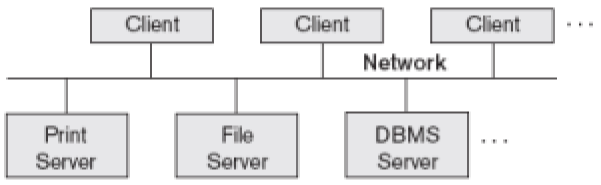
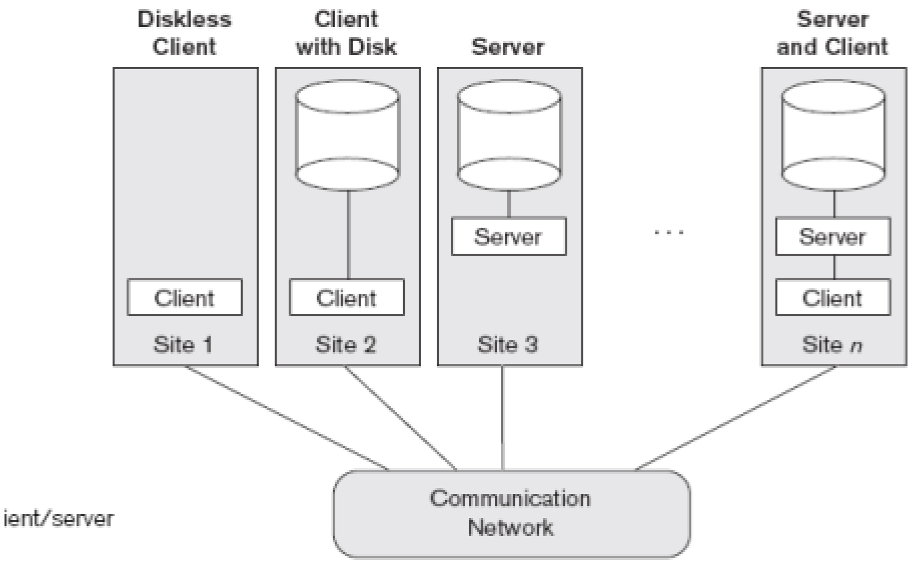


Figure 2.5
Logical two-tier client/server architecture.



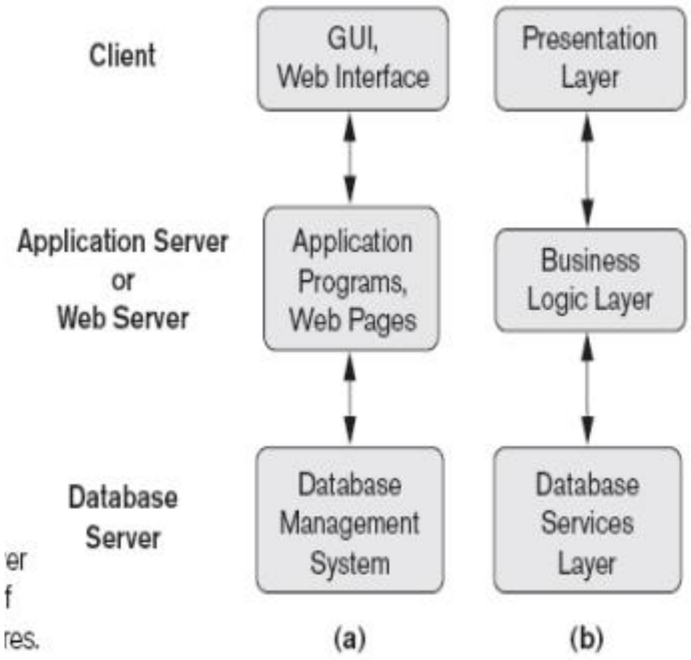
Three-Tier Client/Server Architectures for DBMS

Three tier architecture adds intermediate layer between client and the database server called Application server or Web server improves database security by checking client's credentials before forwarding a request to the database server. User interface, application rules, and data access act as the three tiers.

➢Server -Runs application programs and stores business rules

➢Clients -contain GUIs and application-specific business rules.

➢Intermediate server - Accepts request from the client and process it. Sends database queries and commands to database server . Acts as a channel for passing partially processed data from the database server to clients .Further processed and filtered for GUI



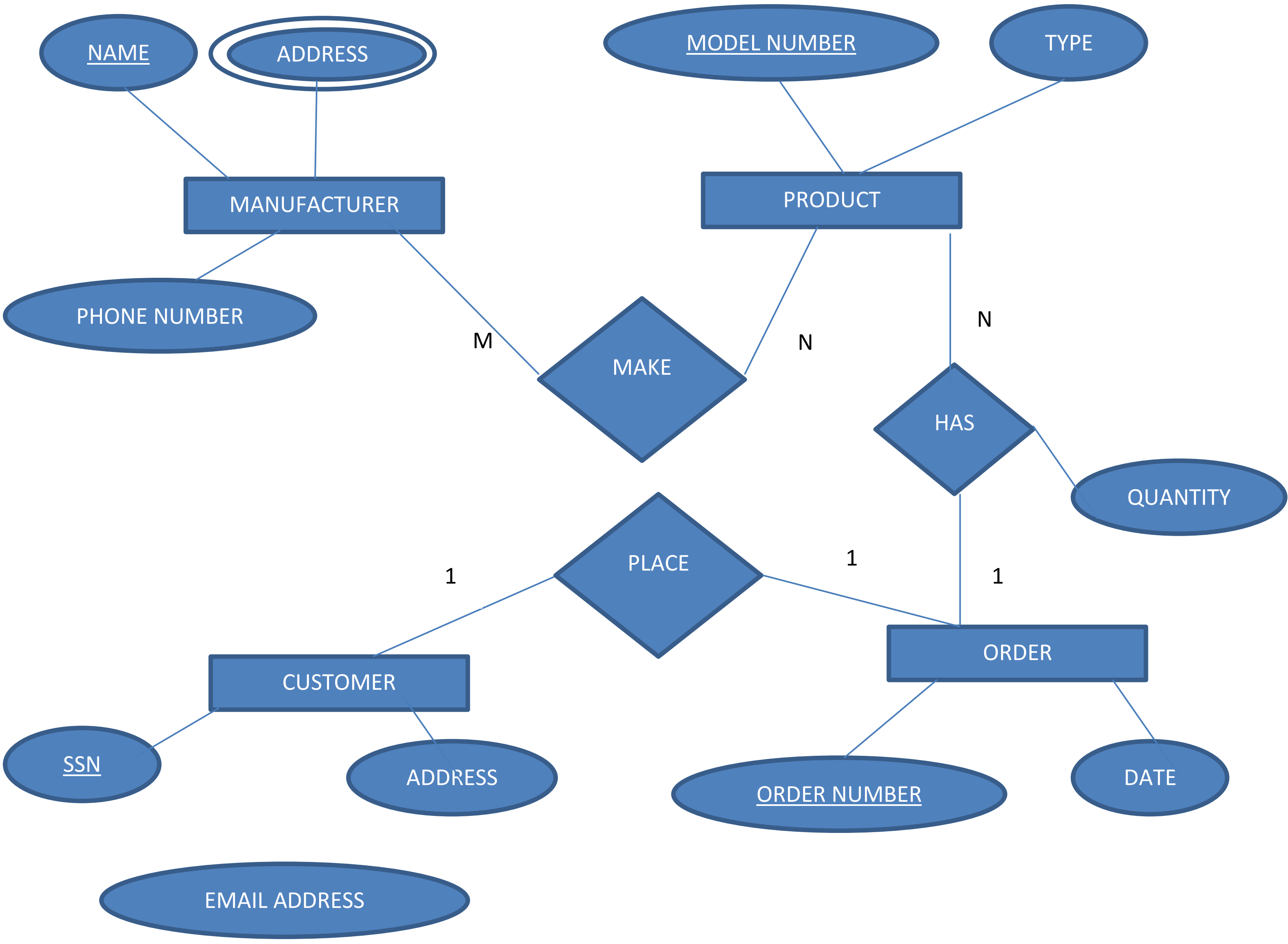
11b) Draw an ER diagram based on the following information,

• Manufacturers have a name, which we may assume is unique, an address, and a phone number

• Products have a model number and a type. Each product is made by one manufacturer, and different manufacturers may have different products with the same model number. However, you may assume that no manufacturer would have two products with the same model number

• Customers are identified by their unique social security number. They have email addresses, and physical addresses. Several customers may live at the same (physical) address, but we assume that no two customers have the same email address

• ***An order has a unique order number, and a date. An order is placed by one customer. For each order, there are one or more products ordered, and there is a quantity for each product on the order.***



12 a) Write briefly about any three types of database end users

Casual end users : occasionally access the database, but they may need different information each time.

Naive or parametric end users :Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates called canned transactions that have been carefully programmed and tested.The tasks that such users perform are varied.

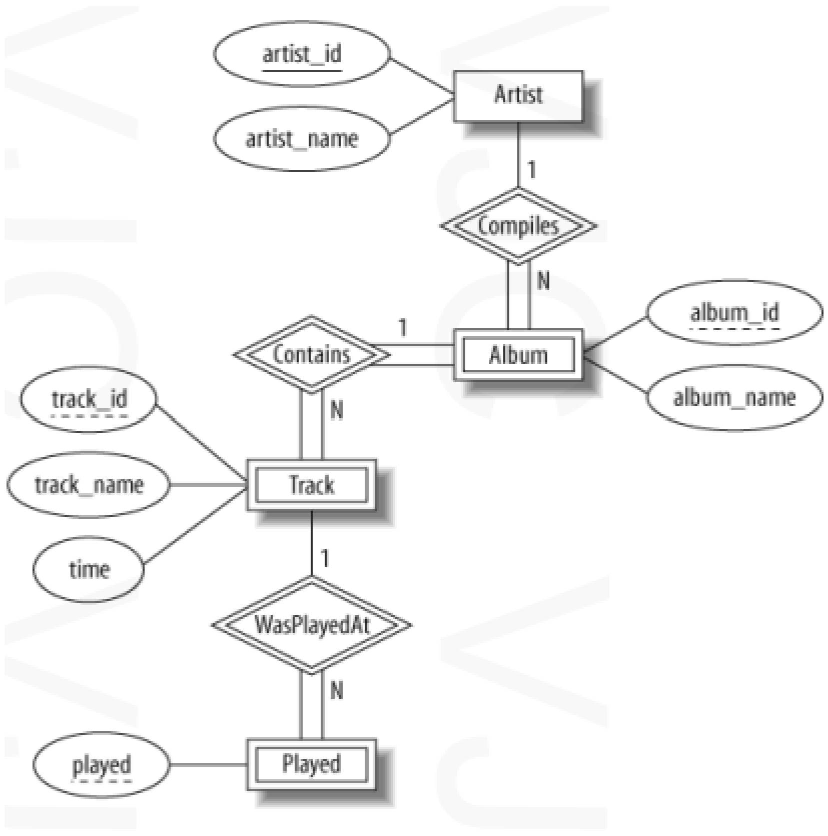
Bank tellers check account balances and post withdrawals and deposits.

Reservation agents for airlines, hotels, and car rental companies check availability for a given request and make reservations.

Sophisticated end users : include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

Standalone users : maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces

b) Interpret the following ER diagram



Entity	Attributes
Artist(Strong entity)	Artist id(Key attribute) Artist name
Album(Weak entity)	Album id(partial key) Album name
Track(Weak entity)	Track id(partial key) Track name Time
Played(Weak entity)	Played(partial key)
Relationship	**Constraints**
Compiles	One artist can compile many albums. It is not necessary that every artist must compile an

	album. Each album must be compiled by some artis Being a weak entity participation of album is total
Contains	One album can contain many tracks All albums must contain tracks. Each track should b some part of some albums. Being a weak entity participation of track is total
Was played at	One track can be played many times. Being a weak entity participation of played is total

13 a) Consider the following schema

Suppliers (sid , sname, address)

Parts (pid, pname, color)

Catalog (sid, pid, cost)

The primary key fields are underlined. Write relational algebra expressions for the following queries:

a) Find the name of parts supplied by supplier with sid=105

$\Pi_{pname}\ (\sigma_{sid=105}\,(parts))$

b) Find the names of suppliers supplying some green part for less than Rs 1000

$\Pi_{sname}\ (\sigma_{color='green'\ AND\ cost<1000}(suppliers*catalog*parts))$

c) Find the IDs of suppliers who supply some red or green

$\Pi_{sid}\ (\sigma_{color='green'\ OR\ color='red'}\,(suppliers*catalog*parts))$

d) Find the names of suppliers who supply some red part

$\Pi_{sname}\ (\sigma_{color='red'}\,(suppliers*catalog*parts))$

13b) Differentiate between the following SQL statements

a) DROP and DELETE

DROP TABLE is used to delete a whole database or just a table.

Syntax:

To drop a table   DROP TABLE <table name>

To drop a database DROP DATABASE <database_name>

The DELETE Statement in SQL is used to delete existing records from a table. We can delete a single record or multiple records depending on the condition we specify in the WHERE clause.

Syntax:

DELETE FROM <table_name> WHERE <condition>;

b) ALTER and UPDATE

The ALTER TABLE statement is used to change the table definition.

Syntax:

To add new column

ALTER TABLE<table name> ADD column name datatype(size);

To add Primary key

ALTER TABLE <table name> ADD PRIMARY KEY(column name);

To add Foreign key

ALTER TABLE <table name> ADD FOREIGN KEY (column name) REFERENCES <table name> (column name);

To remove a Foreign key

ALTER TABLE <table name> DROP <foreign key name>;

To modify the existing columns in a table

ALTER TABLE <table name> MODIFY column_name column_type;

To rename a table

```
	ALTER TABLE <table name> RENAME TO <new table name>
```

UPDATE allows us to change some values in a tuple without necessarily changing all.

We can update single columns as well as multiple columns using UPDATE statement as per our requirement.

Syntax: UPDATE <table_name> SET column1 = value1, column2 = value2, ...WHERE <condition>;

14 a) Write SQL DDL statements based on the following database schema (Assume suitable domain types):

Employee (eid, name, designation, salary, comp_id)

Company (comp_id, cname, address, turnover)

a) Create the above mentioned tables assuming each company has many employees. Mention the primary key, foreign key and not null constraints.

```
CREATE TABLE employee
	( eid VARCHAR(10) PRIMARY KEY,
name VARCHAR(10) NOT NULL,
designation VARCHAR(10) NOT NULL,
	salary INTEGER NOT NULL,
	comp_id VARCHAR(10) NOT NULL);
CREATE TABLE company
	(  comp_id VARCHAR(10) NOT NULL,
	cname VARCHAR(10) NOT NULL,
address VARCHAR(20) NOT NULL,
```

turnover INTEGER NOT NULL,

FOREIGN KEY comp_id REFERENCES employee (comp_id));

b) Insert values into both the tables. Mention in which order insertions will be carried out.

INSERT INTO employee VALUES(‘e1’,’John’,’Supervisor’,10000,’co1’);

INSERT INTO employee VALUES(‘e2’,’Joy’,’Supervisor’,10000,’co2’);

INSERT INTO company VALUES(‘co1’,’company1’,’xyz’,100000);

INSERT INTO company VALUES(‘co2’,’company2’,’abc’,200000);

c) Modify the table Employee to include a new column “years_of_exp”

ALTER TABLE employee ADD COLUMN years_of_exp INTEGER;

d) Increment the salary of employees whose salary is less than Rs25000 by 5%

UPDATE employee SET salary=salary*1.05 WHERE salary<25000

14 b) Illustrate any three ways of using INSERT statement in SQL.

For the employee(fname,lname,dno,ssn) insert statements can be

a) INSERT INTO employee

VALUES(‘john’,’david’,1,’abc123’);

b) INSERT INTO employee(fname,lname,dno,ssn) VALUES(‘john’,’david’,1,’abc123’);

c) INSERT INTO employeedepartment(dnum,essn)

SELECT dno,ssn

FROM employee;

15 a) For the relation schema below, give an expression in SQL for each of the queries that follows:

employee (ID, person_name, street, city)

works (ID, company_name, salary)

company (company_name, city)

manages (ID, manager_id)

a) Find the employees whose name starts with 'C'

```
SELECT person_name
FROM employee
WHERE person_name
 LIKE 'C%'
```

b) Find the name of managers of each company

```
SELECT person_name
FROM employee E,manages M
WHERE  M. manager_id=E.ID
```

c) ***Find the ID, name, and city of residence of employees who works for "First*** Bank Corporation" and earns more than Rs50000

```
SELECT ID, person_name,  city
FROM employee E,works W
WHERE E.ID=W.ID
AND company_name="First Bank Corporation"
AND salary>50000
```

d) Find the name of companies whose employees earn a higher salary, on average, than the average salary at "First Bank Corporation"

SELECT company-name

FROM works

WHERE salary >

ALL (SELECT AVG(salary) FROM works

WHERE company-name = ' ***First*** Bank Corporation')

15 b) Differentiate correlated and non-correlated nested queries with suitable examples

Whenever a condition in the WHERE-clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated.

Correlated- Retrieve the name of each employee who has a dependent with the same first name as the employee

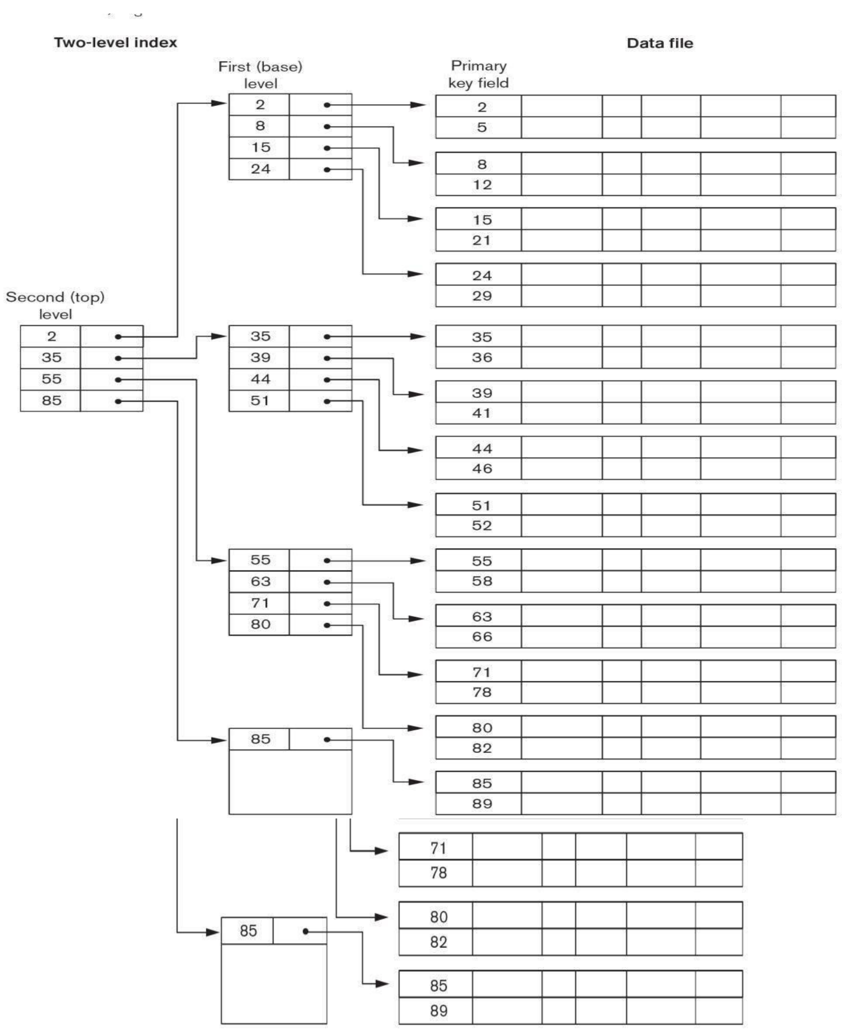
```
SELECT    E.FNAME, E.LNAME
 FROM          EMPLOYEE AS E
 WHERE    E.SSN IN (SELECT     ESSN
          FROM      DEPENDENT
          WHERE     ESSN=E.SSN AND
                E.FNAME=DEPENDENT_NAME)
```

Non-Correlated -Retrieve the name and address of all employees who work for the 'Research' Department

```
SELECT FNAME,LNAME, ADDRESS
FROM EMPLOYEE
WHERE DNO IN (SELECT DNUMBER
                        FROM DEPARTMENT
                        WHERE DNAME='Research')
```

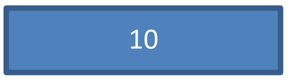
16 a) What is multi-level indexing? How does it improve the efficiency of searching an index file?

Because a single-level index is an ordered file, we can create a primary index to the index itself;In this case, the original index file is called the first- level index and the index to the index is called the second-level index.We can repeat the process, creating a third, fourth, ..., top level until all entries of the top level fit in one disk block.A multi-level index can be created for any type of first- level index (primary, secondary, clustering) as long as the first-level index consists of more than one disk block.Given index field value multi- level indexing reduces no of block acess while searching

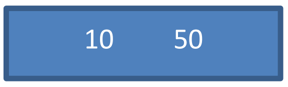


16 b) Insert the following keys, in the order given, into a B -tree of order 3: {10, 50, 20, 5, 22, 25}

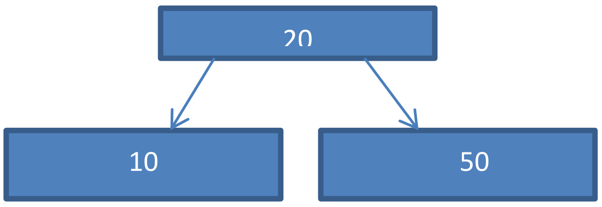
Inserting 10



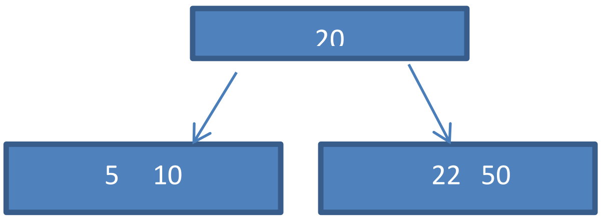
Inserting 50



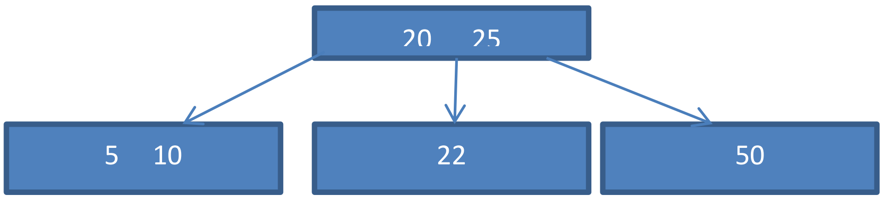
Inserting 20



Inserting 5,22



Inserting 25



17 a) Consider a relation R(A, B, C, D, E) with FDs AB → C, AC → B, BC → A, D → E. Determine all the keys of relation R. Also decompose the relation into collections of relations that are in BCNF.

Key=ABD - since Closure of ABD=(A, B, C, D, E)

Decomposing into BCNF

<u>A</u> <u>B</u> C (FDs AB → C, AC → B, BC → A)

<u>D</u> E (FDs D → E)

17 b) Write briefly on the different types of anomalies in designing a database.

Insertion Anomaly

Consider the relation:

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

We cannot insert a project unless an employee is assigned to it. Conversely we cannot insert an employee unless an he/she is assigned to a project.

Deletion Anomaly

Consider the relation:

EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)

If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database.

Modification Anomaly

EMP_DEPT, if we change the value of one of the attributes of a particular department say,the manager of department 5 we must update the tuples of all employees who work in that department;otherwise, the database will become inconsistent.If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

18 a) Consider a relation schema R (A,B,C,D) with the following functional dependencies A → B, B → C, C → D, D → B. Determine whether the decomposition of R into R1 (A , B) , R2 (B , C) and R3 (B , D) is lossless or lossy. Write the complete steps.

Representing decompositions and attributes as matrix

A	***B***	***C***	***D***
A1	***A2***		
	A2	***A3***	
		A3	***A4***

A	***B***	***C***	***D***
A1	***A2***	***A3*** ***(Since A2→A3)***	***A4*** ***(Since A3→A4)***
	A2	***A3***	
		A3	***A4***

the decomposition of R into R1 (A , B) , R2 (B , C) and R3 (B , D) is lossless

18 b) What is dependency preservation property for decomposition? Why is it important?

The dependency preservation property says that after decomposing a relation R into R1 and R2, all dependencies of the original relation R must be present either in R1 or R2 or they must be derivable using the combination of functional dependencies present in R1 and R2. An ideal decomposition should be lossless join decomposition and dependency preserving.

Decomposition helps to remove anomalies, redundancy, and other problems in a DBMS

19 a) Explain briefly the ACID properties of a transaction.

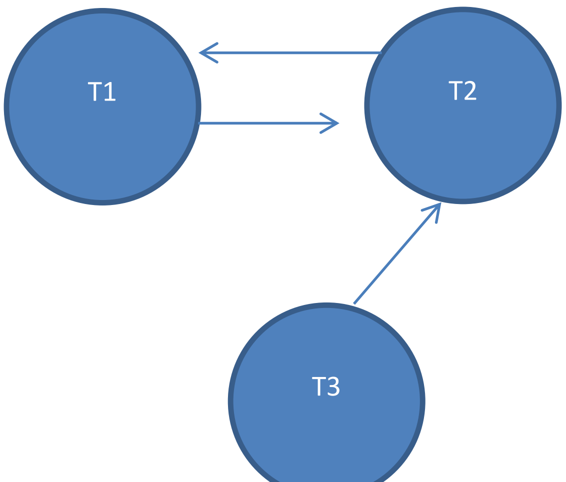
Atomicity -A transaction is said to be atomic if either all of the commands are succeeded or none of them

Consistency -A transaction should be consistency preserving • It should take the database from one consistent sate to another

Isolation - Transactions should be isolate to each other during concurrent execution **Durability** or Permanency - The changes applied to the database by committed transaction must persist in the database.these changes must not be lost because of any failure

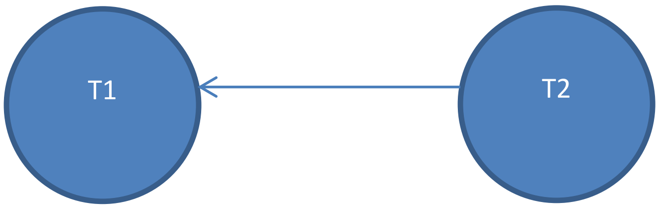
19 b) Check whether the given schedules are conflict serializable or not

i) S1 : R1(X) , R2(X) , R1(Y) , R2(Y) , R3(Y) , W1(X) , W2(Y)



Since there is loop in precedence graph schedule is not conflict serializable

ii) S2 : R1(X) , R2(X) , R2(Y) , W2(Y) , R1(Y) , W1(X)



Since there is no loop in precedence graph schedule is conflict serializable

20 a) What is two phase locking protocol? How does it guarantee serializability?

If every individual transactions follows 2PL, then all schedules in which these transactions participate become Conflict Serializable . 2PL Schedule are always Conflict Serializable Schedules A schedule is said to be in 2PL if all transactions perform locking and unlocking in 2 phases

1. Growing Phase: ▫ New locks on data items can be acquired but none can be unlocked ▫ Only locking is allowed no unlocking

2. Shrinking Phase ▫ Existing locks may be released but no new locks can be acqired ▫ Only unlocking is allowed no locking

Consider the schedule

T1	T2
Write lock(A)	
Read(A)	

Here T1 starts execution performing Read(A) ,Write(A) .When T1 unlocks A ,T2 will start.After completing Read(A) in T2, Read(B) of T1 will be performed.So the order of execution is T1->T2->T1which is not serial

Write(A)	
Unlock(A)	
	Read lock(A)
	Read(A)
Read lock(B)	
Read(B)	

T1	T2
Write lock(A)	
Read(A)	
Write(A)	
	Read lock(A)
	Read(A)
Read lock(B)	
Read(B)	
Unlock(A)	

Here T1 starts execution performing Read(A) ,Write(A).T2 will request for lock but needs to wait until unlock. .So the order of executionis T1->T2 which is serial

20 b) What are the main characteristics of NOSQL systems in the areas related to data models and query languages?

NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data. Unlike traditional relational databases that use tables with pre-defined schemas to store data, NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data.

The term NoSQL originally referred to “non-SQL” or “non-relational” databases, but the term has since evolved to mean “not only SQL,” as NoSQL databases have expanded to include a wide range of different database architectures and data models.

NoSQL databases are generally classified into four main categories:

- Document databases: These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.
- Key-value stores: These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.
- Column-family stores: These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data.
- Graph databases: These databases store data as nodes and edges, and are designed to handle complex relationships between data.

NoSQL databases are often used in applications where there is a high volume of data that needs to be processed and analyzed in real-time, such as social media analytics, e-commerce, and gaming. They can also be used for other applications, such as content management systems, document management, and customer relationship management.However, NoSQL databases may not be suitable for all applications, as they may not provide the same level of data consistency and transactional guarantees as traditional relational databases.

NoSQL originally referring to non SQL or non relational is a database that provides a mechanism for storage and retrieval of data. This data is modeled in means other than the tabular relations used in relational databases. NoSQL databases are used in real-time web applications and big data and their use are increasing over time.

NoSQL systems are also sometimes called Not only SQL to emphasize the fact that they may support SQL-like query languages. A NoSQL database includes simplicity of design, simpler horizontal scaling to clusters of machines and finer control over availability. The data structures used by NoSQL databases are different from those used by default in relational databases which makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it should solve.

Key Features of NoSQL :

- **Dynamic schema:** NoSQL databases do not have a fixed schema and can accommodate changing data structures without the need for migrations or schema alterations.
- **Horizontal scalability:** NoSQL databases are designed to scale out by adding more nodes to a database cluster, making them well-suited for handling large amounts of data and high levels of traffic.
- **Document-based:** Some NoSQL databases, such as MongoDB, use a document-based data model, where data is stored in semi-structured format, such as JSON or BSON.
- **Key-value-based:** Other NoSQL databases, such as Redis, use a key-value data model, where data is stored as a collection of key-value pairs.
- **Column-based:** Some NoSQL databases, such as Cassandra, use a column-based data model, where data is organized into columns instead of rows.
- **Distributed and high availability:** NoSQL databases are often designed to be highly available and to automatically handle node failures and data replication across multiple nodes in a database cluster.
- **Flexibility:** NoSQL databases allow developers to store and retrieve data in a flexible and dynamic manner, with support for multiple data types and changing data structures.
- **Performance:** NoSQL databases are optimized for high performance and can handle a high volume of reads and writes, making them suitable for big data and real-time applications.

Advantages of NoSQL.

- **High scalability :** NoSQL databases use sharding for horizontal scaling. Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding. Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement but horizontal scaling is easy to implement. Examples of horizontal scaling databases are MongoDB, Cassandra, etc. NoSQL can handle a huge amount of data because of scalability, as the data grows NoSQL scale itself to handle that data in an efficient manner.
- **Flexibility:** NoSQL databases are designed to handle unstructured or semi-structured data, which means that they can accommodate dynamic changes to the data model. This makes NoSQL databases a good fit for applications that need to handle changing data requirements.
- **High availability :** Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.
- **Scalability:** NoSQL databases are highly scalable, which means that they can handle large amounts of data and traffic with ease. This makes them a good fit for applications that need to handle large amounts of data or traffic
- **Performance:** NoSQL databases are designed to handle large amounts of data and traffic, which means that they can offer improved performance compared to traditional relational databases.
- **Cost-effectiveness:** NoSQL databases are often more cost-effective than traditional relational databases, as they are typically less complex and do not require expensive hardware or software.

Types of NoSQL database:

- **Graph Databases**: Examples – Amazon Neptune, Neo4j
- **Key value store:** Examples – Memcached, Redis, Coherence
- **Tabular:** Examples – Hbase, Big Table, Accumulo
- **Document-based:** Examples – MongoDB, CouchDB, Cloudant