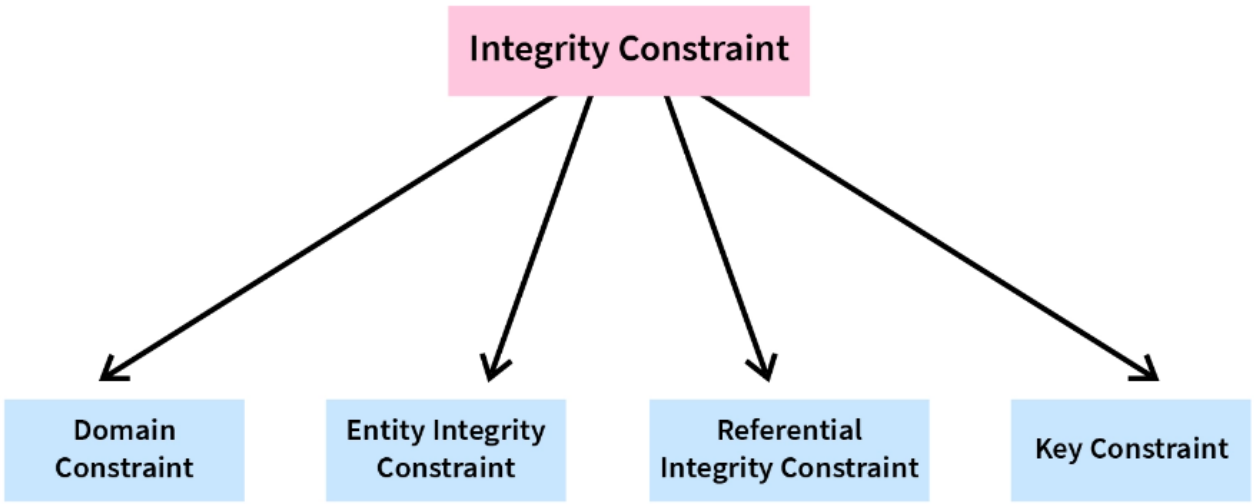
Define various integrity constraints in relational model.



1. Domain constraints

- Domain constraint is defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- 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

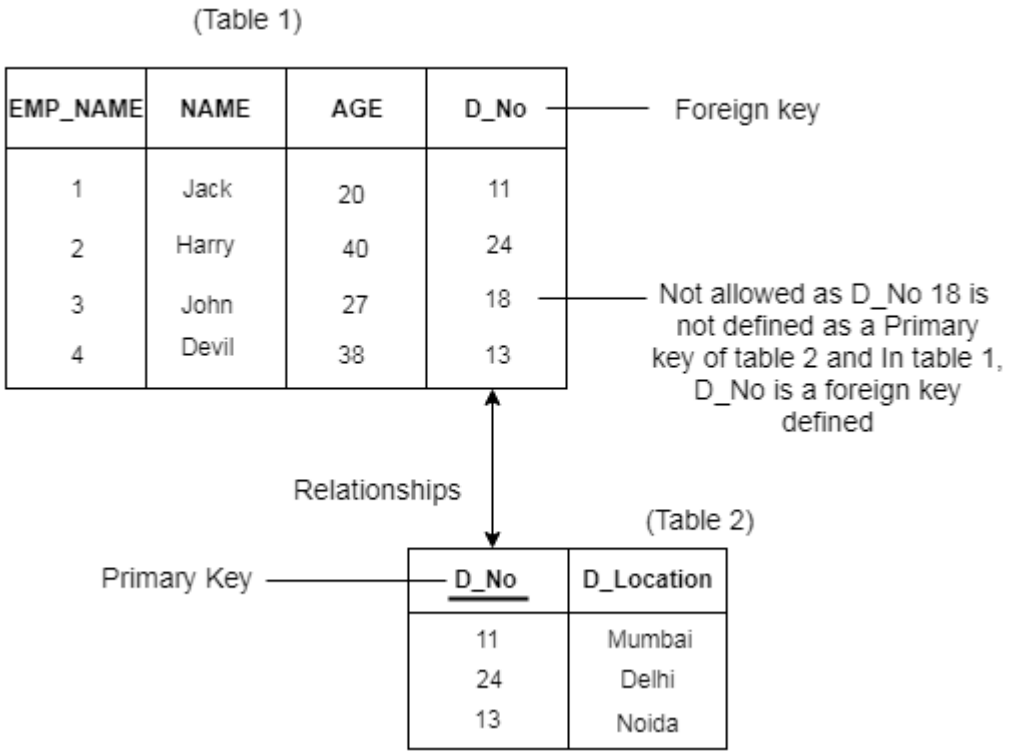
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:



4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

5. Semantic constraints

These constraints cannot be directly expressed in the schemas of the data model, and hence they must be expressed and enforced by the application programs. Examples of such constraints are the salary of an employee should not exceed the salary of the employee's supervisor. In SQL, CREATE ASSERTION and CREATE TRIGGER statements are used to enforce semantic constraints.

What do you mean by canonical cover of a set of functional dependencies F over a relation schema R?

A canonical cover of a set of functional dependencies F is a simplified set of functional dependencies that has the same closure as the original set F.

A canonical cover F_c of a set of functional dependencies F satisfies the following properties:

- F logically implies all dependencies in F_c.
- F_c logically implies all dependencies in F.
- No functional dependency in F_c contains an extraneous attribute.
- Each left side of a functional dependency in F_c is unique. That is, there are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in such that $\alpha_1 \rightarrow \alpha_2$.

What are conflict operations in concurrency control?

Two operations are said to be conflicting if all conditions are satisfied:

- They belong to different transactions
- They operate on the same data item
- At Least one of them is a write operation

Example:

- **Conflicting** operations pair (R1(A), W2(A)) because they belong to two different transactions on the same data item A and one of them is a write operation.
- Similarly, (W1(A), W2(A)) and (W1(A), R2(A)) pairs are also **conflicting**.
- On the other hand, the (R1(A), W2(B)) pair is **non-conflicting** because they operate on different data items.
- Similarly, ((W1(A), W2(B)) pair is **non-conflicting.**

Distributed Database Transparencies

In Distributed Database Management System, there are four types of transparencies, which are as follows –

1. **Transaction Transparency:**
   - Ensures that distributed transactions maintain integrity and atomicity.
   - Complexity arises due to fragmentation, allocation, and replication structures.
2. **Performance Transparency:**
   - Requires the DDBMS to perform similarly to a centralized DBMS in terms of performance.
   - Requires efficient distributed query processing while considering fragmentation, replication, and allocation structures.
3. **DBMS Transparency:**
   - Applicable to heterogeneous DDBMS, hides differences in local DBMS.
   - Complexity arises from the need to generalize across different systems.
4. **Distribution Transparency:**
   - Presents the distributed database as a single logical entity to users.
   - Includes various types:
     - Fragmentation Transparency: Users do not need to know about fragmented data.
     - Location Transparency: Users know how data is fragmented but not its location.
     - Replication Transparency: Users are unaware of replicated data fragments.
     - Local Mapping Transparency: Users define fragment names and data locations, considering any duplications.
     - Naming Transparency: Ensures unique object names across sites, handled by central name server or site identifiers.

What do you mean by distributed query processing? How is semi- join operation helpful in it?

A distributed database query is processed in following stages:

- **Query Mapping:** Translate input query to global schema algebraically, without considering data distribution or replication. It involves normalization, semantic analysis, simplification, and restructuring.
- **Localization:** Map distributed query onto individual fragments at different sites, considering data distribution and replication. Generate separate queries for each fragment.
- **Global Query Optimization:** Select most efficient strategy from candidate queries. Permute operation order within fragment queries. Consider CPU, I/O, and communication costs; prioritize communication costs in WAN. Total cost is weighted combination of these costs.
- **Local Query Optimization:** This stage is common to all sites in the distributed database. The techniques are similar to those used in centralized systems.

Using Semi-Join in Distributed Query Processing

The idea is to send the joining column of one relation R to the site where the other relation S is located; this column is then joined with S. Following that, the join attributes, along with the attributes required in the result, are projected out and shipped back to the original site and joined with R.

Explain nested and long duration transactions with suitable examples.

Nested transactions refer to transactions that are embedded within other transactions. The outer transaction is known as the parent transaction, and the inner transactions are called child transactions. Child transactions can have their own operations and changes to the database, and they can commit or abort independently of the parent transaction. If a child transaction commits successfully, its changes are not visible to the parent transaction until the parent commits. If the child aborts, its changes are rolled back, and the parent can decide whether to commit or abort based on the outcome of the child transaction.

Example:

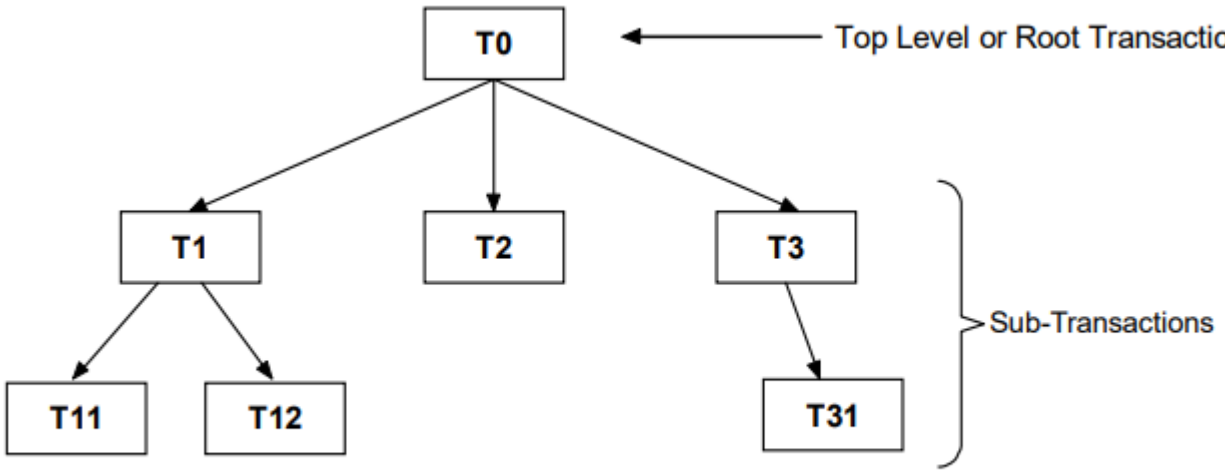
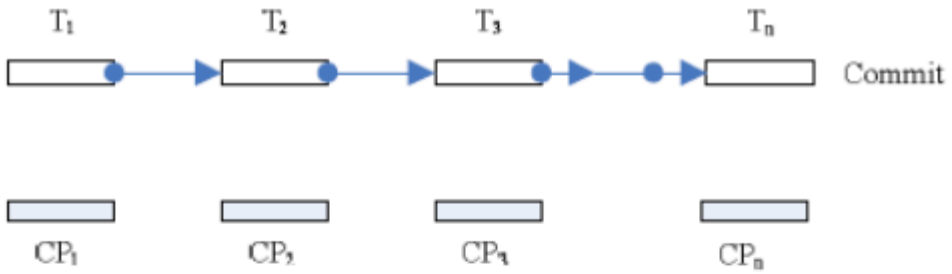


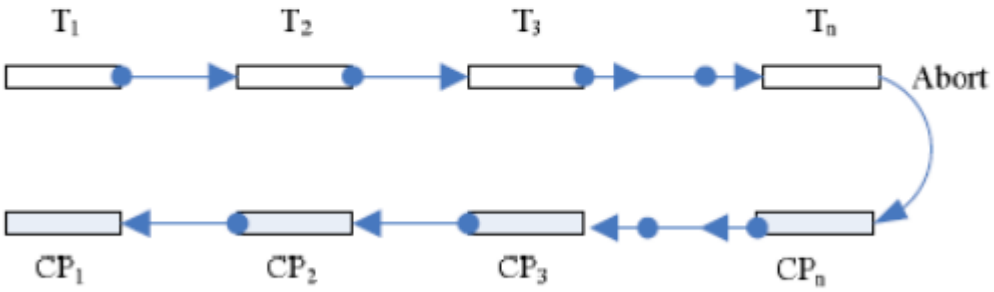
Figure 17: Nested Transaction Model

Long duration transaction (also called Saga Transaction) makes use of the concept of compensating transactions to support transactions whose execution time is long. A Sagas transaction consists of a consecutive chain of flat transactions S_i that can commit independently. For each flat transaction S_i, there is a compensating transaction CP_i that will undo the effect of the transaction S_i if the transaction S_i aborts. A compensating transaction CP_i in the Sagas chain is triggered by the associated transaction S_i or the compensating transaction CP_{i+1}. If the Sagas transaction commits, no compensating transaction CP_i is initiated, otherwise the chain of compensating transactions is triggered.

Example:



a : A successful Sagas



b: An unsuccessful Sagas

Figure 19: Successful and Unsuccessful Sagas

What is the difference between object relational model and object oriented model?

Feature	Object-Oriented Model	Object-Relational Model
Data Representation	Encapsulated data within objects	Data stored in tables with complex data types
Relationships	Object references and inheritance	Foreign key constraints
Focus	Software development	Data persistence and retrieval
Performance	Potentially slower due to object overhead	Faster for large datasets and complex queries
Examples	Java, C++, Python	PostgreSQL, Oracle, DB2

Write difference between persistent and transient objects. Explain the term object versioning.

Feature	Persistent Object	Transient Object
Lifetime	Outlives the program that created it, existing on external storage (e.g., database)	Exists only in memory while the program runs, dying when the program terminates
Data Storage	Stored in databases, files, or other external sources	Stored in memory using variables and data structures

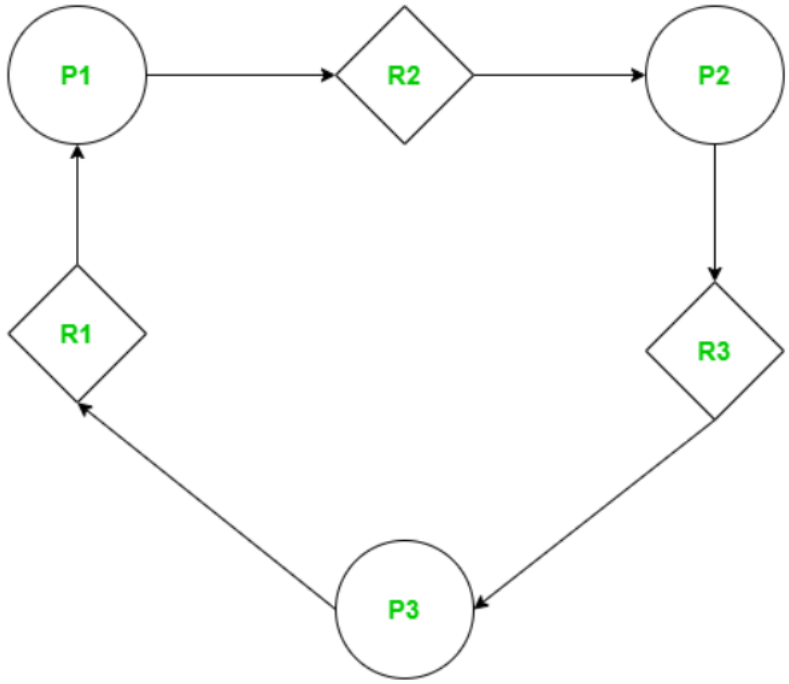
Access	Can be accessed by different programs or sessions after creation	Only accessible by the program that created it
Durability	Changes are preserved between program runs	Changes are lost when the program terminates
Examples	User accounts, product data, transaction logs	Temporary variables, calculations, function arguments

Object versioning refers to maintaining multiple historical versions of a persistent object. This allows tracking changes over time, reverting to previous states if needed, and supporting features like audit logs or rollbacks.

Explain the phantom deadlock concept in distributed database with suitable example?

Phantom deadlocks, in distributed systems, refer to situations where multiple processes or threads get stuck and can't move forward because of conflicts in synchronization and resource allocation. This happens when different tasks are being executed at the time causing each process to wait for a resource that is held by another process. This creates a dependency that brings the system to a halt.

Example:



In this diagram,

- The arrows show the relationships of waiting.
- For instance, P1 is waiting for R2. P2 is waiting for R3.
- There seems to be a pattern of waiting; P1 is waiting for R2 P2 is waiting for R3 and P3 is waiting for R1.
- This circular pattern suggests a deadlock.

In this WFG representation of the workflow we have three processes (P1, P2, P3) and three resources (R1, R2, R3). The arrows indicate the relationships where processes are waiting for resources or holding them.

- **Process P1 and Resource R2:** Process P1 is currently waiting for Resource R2. This implies that P1 has made a request, for R2 and cannot proceed until it obtains it. The reason behind this waiting situation could be resource unavailability or the need for synchronization.
- **Resource R2 and Process P2:** Process P2 is currently holding Resource R2. This indicates that P2 has acquired control over R2 causing other processes like P1 to wait until it is released.
- **Resource R2 and Resource R3:** There is also a waiting relationship between Resource R2 and Resource R3. This suggests that having access to R2 is a prerequisite, for utilizing or depending on R3 creating a chain dependency.
- **Resource R3 and Process P2:** Process P2 is currently waiting for Resource R3. This implies that Process P2 is currently, in a state of pause as it awaits the availability of Resource R1. Similar to instances where processes are waiting P2s advancement is put on hold until it successfully obtains access to R1.
- **Resource R1 and Process P3:** At present Process P3 is also in a state of waiting as it awaits the availability of Resource R1. This forms a cycle, where P3s wait for R1 connects back, to P1s wait for R2 resulting in a pattern of dependencies.

Phantom deadlocks arise when these logical dependencies cause processes to wait indefinitely even if there is no blocking of resources as seen in deadlocks.

What is distributed deadlock handling? Suggest one method of it.

Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T in the set. Hence, each transaction in the set is in a waiting queue, waiting for one of the other trans actions in the set to release the lock on an item. But because the other transaction is also waiting, it will never release the lock.

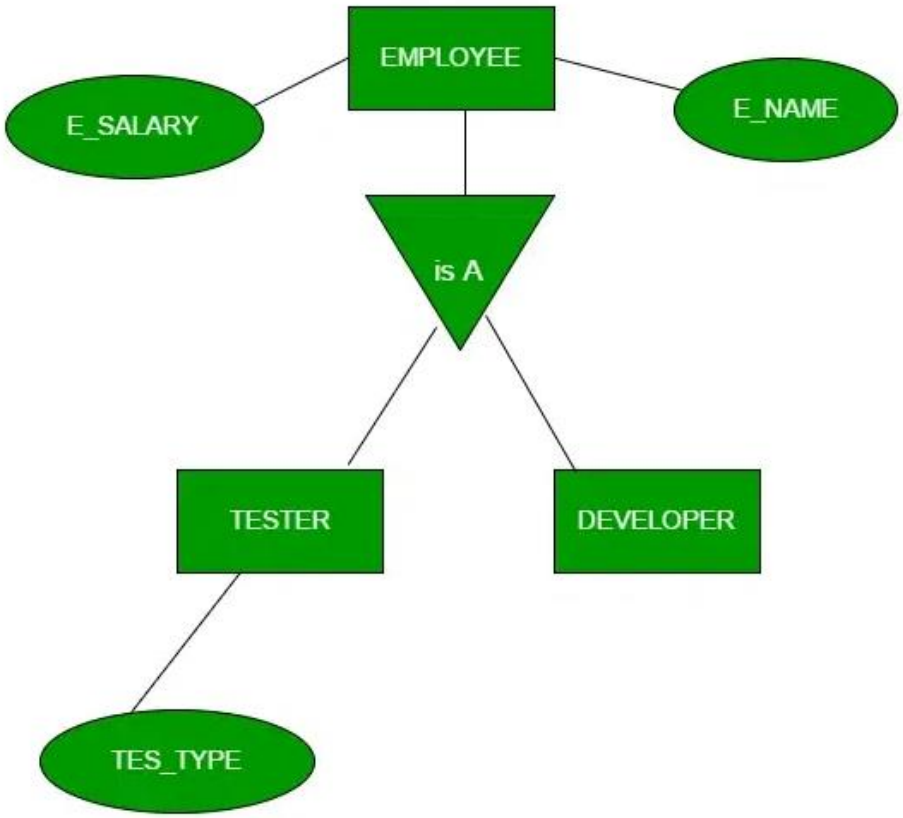
Deadlock Prevention: No waiting algorithm: If a transaction is unable to obtain a lock, it is immediately aborted and then restarted after a certain time delay without checking whether a deadlock will actually occur or not. In this case, no transaction ever waits, so no deadlock will occur.

Deadlock Detection: Wait-for graph: One node is created in the wait-for graph for each transaction that is currently executing. Whenever a transaction T_i is waiting to lock an item X that is currently locked by a transaction T_j , a directed edge ($T_i \rightarrow T_j$) is created in the wait-for graph. When T_j releases the lock(s) on the items that T_i was waiting for, the directed edge is dropped from the wait-for graph. We have a state of dead lock if and only if the wait-for graph has a cycle.

Explain the concept of specialization used in extended ER model. What are various types of possible specializations?

In specialization, an entity is divided into sub-entities based on its characteristics. It is a top-down approach where the higher-level entity is specialized into two or more lower-level entities.

Example:



Specialization

For Example, an EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER, etc. In this case, common attributes like E_NAME, E_SAL, etc. become part of a higher entity (EMPLOYEE), and specialized attributes like TES_TYPE become part of a specialized entity (TESTER).

Various Types of Specializations in ER Model:

- **Total Disjoint Specialization:** Each instance of the superclass must be a member of exactly one subclass.
- **Partial Disjoint Specialization:** Some instances of the superclass might not belong to any subclass.
- **Total Overlapping Specialization:** Each instance of the superclass must be a member of at least one subclass, and it can belong to multiple subclasses.
- **Partial Overlapping Specialization:** Instances of the superclass may belong to one or more subclasses, but it is not mandatory for every instance to belong to any subclass.

Can an entity set have multiple specializations?

Yes, an entity set can have multiple specializations. This means that a single entity set can serve as a superclass for multiple subclass hierarchies or participate in several specialization relationships simultaneously.

Explain Semi-join, Equi-Join and Outer -join operations in relational algebra with suitable example.

Semi-join is a type of join that is applied to relations to join them based on the related columns. When semi-join is applied, it returns the rows from one table for which there are matching records in another related table.

Characteristics:

- A Semi-join returns rows from the left table for which there are corresponding matching rows in the right table.
- Unlike regular joins which include the matching rows from both tables, a semi-join only includes columns from the left table in the result.

Syntax:

using `**EXISTS**`:

SELECT column1, column2, …FROM table1
WHERE EXISTS (SELECT 1 FROM table2 WHERE table1.column = table2.column);

using `**IN**`:

SELECT column1, column2, …FROM table1WHERE column IN (SELECT column FROM table2);

Example:

Customers table:

Customer_ID	Customer_Name
01	Alice
02	Bob
03	Charlie
04	David

Orders Table

Customer_ID	Order_ID	Order_Name
02	101	Stationery
01	102	Books
04	103	Pens

Query for Semi-Join:

```
SELECT Customers.Customer_ID, Customers.Customer_Name
FROM Customers
WHERE EXISTS (
  SELECT 1
  FROM Orders
  WHERE Customers.Customer_ID = Orders.Customer_ID
);
```

Output:

Customer_ID	Customer_Name
01	Alice
02	Bob
04	David

https://www.youtube.com/watch?v=jC1AcjOpmIY

https://www.youtube.com/watch?v=ifiP9nbZF4I

How size estimation of NATURAL JOIN operation (between relations R & S) is performed?

Estimating the size of a NATURAL JOIN operation between two relations \(R \) and \(S \) involves understanding the number of common attributes between the relations and estimating the resulting cardinality of the joined relation.

Steps for estimating size of Natural Join operation:

1. **Identify Common Attributes:** Determine the common attributes between relations R and S. These are the attributes used to join the relations in the NATURAL JOIN operation.

2. **Estimate Cardinality:** Estimate the number of rows in the resulting joined relation. This can be done by examining the data and understanding the relationship between the common attributes in R and S.

3. **Calculate Total Size:** Once the cardinality is estimated, you can calculate the total size of the resulting joined relation by considering the size of each tuple, which includes the combined size of attributes from both R and S.

SELECT *

FROM R

NATURAL JOIN S;

Explain various types of data fragmentation methods used in distributed database with suitable example.

We have three methods for data fragmenting of a table:

- **Horizontal fragmentation**
- **Vertical fragmentation**
- **Mixed or Hybrid fragmentation**

Horizontal fragmentation

Horizontal fragmentation refers to the process of dividing a table horizontally by assigning each row (or a group of rows) of relation to one or more fragments. These fragments can then be assigned to different sites in the distributed system. Some of the rows or tuples of the table are placed in one system and the rest are placed in other systems. The rows that belong to the horizontal fragments are specified by a condition on one or more attributes of the relation. In relational algebra horizontal fragmentation on table T, can be represented as follows:

$\sigma_p(T)$
where, σ is relational algebra operator for selection

p is the condition satisfied by a horizontal fragment

For example, consider an EMPLOYEE table (T):

Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1
103	C	abc	5500	2
104	D	abc	5000	2

105	E	abc	2000	2

This EMPLOYEE table can be divided into different fragments like:

EMP 1 = $\sigma_{Dep = 1}$ EMPLOYEE
EMP 2 = $\sigma_{Dep = 2}$ EMPLOYEE
These two fragments are: T1 fragment of Dep = 1

Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1

Similarly, the T2 fragment on the basis of Dep = 2 will be :

Eno	Ename	Design	Salary	Dep
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

Now, here it is possible to get back T as **T = T1 ∪ T2 ∪ …. ∪ TN**

Vertical Fragmentation

Vertical fragmentation refers to the process of decomposing a table vertically by attributes or columns. In this fragmentation, some of the attributes are stored in one system and the rest are stored in other systems. This is because each site may not need all columns of a table. In order to take care of restoration, each fragment must contain the primary key field(s) in a table. The fragmentation should be in such a manner that we can rebuild a table from the fragment by taking the natural JOIN operation and to make it possible we need to include a special attribute called ***Tuple-id*** to the schema. For this purpose, a user can use any super key. And by this, the tuples or rows can be linked together. The projection is as follows:

$\pi_{a1, a2, \ldots, an}(T)$
where, π is relational algebra operator

a1…., an are the attributes of T

T is the table (relation)

For example, for the EMPLOYEE table we have T1 as :

Eno	Ename	Design	Tuple_id
101	A	abc	1
102	B	abc	2
103	C	abc	3
104	D	abc	4
105	E	abc	5

For the second. sub table of relation after vertical fragmentation is given as follows :

Salary	Dep	Tuple_id
3000	1	1
4000	2	2
5500	3	3
5000	1	4
2000	4	5

This is T2 and to get back to the original T, we join these two fragments T1 and T2 as $\Pi_{EMPLOYEE}(T1 \bowtie T2)$

Mixed Fragmentation

The combination of vertical fragmentation of a table followed by further horizontal fragmentation of some fragments is called **mixed or hybrid fragmentation**. For defining this type of fragmentation we use the SELECT and the PROJECT operations of relational algebra. In some situations, the horizontal and the vertical fragmentation isn't enough to

distribute data for some applications and in that conditions, we need a fragmentation called a mixed fragmentation.
Mixed fragmentation can be done in two different ways:

1. The first method is to first create a set or group of horizontal fragments and then create vertical fragments from one or more of the horizontal fragments.
2. The second method is to first create a set or group of vertical fragments and then create horizontal fragments from one or more of the vertical fragments. The original relation can be obtained by the combination of JOIN and UNION operations which is given as follows:

$\sigma_P(\pi_{a1, a2..,an}(T))$

$\pi_{a1,a2....,an}(\sigma_p(T))$

For example, for our **EMPLOYEE** table, below is the implementation of mixed fragmentation is $\pi_{Ename, Design}(\sigma_{Eno < 104}(EMPLOYEE))$
The result of this fragmentation is:

Ename	Design
A	abc
B	abc
C	abc

What is the role of object identity in Object Oriented data model? How persistence is handled in an OO database system?

Object identifier(OID) is used by the system to identify each object uniquely and to create and manage inter-object references. The value of an OID is not visible to the external user. The main property required of an OID is that it be immutable; that is, the OID value of a particular object should not change. Each OID must be used only once; that is, even if an object is removed from the data base, its OID should not be assigned to another object.

Handling persistence in OO database system:

- **Naming:** The naming mechanism involves giving an object a unique persistent name within a particular database. This persistent object name can be given via a specific statement or operation in the program. The named persistent objects are used as entry points to the database through which users and applications can start their database access.

- **Reachability:** The reachability mechanism works by making the object reachable from some other persistent object. An object B is said to be reachable from an object A if a sequence of references in the database lead from object A to object B.

Explain following terminology in transaction management-

a) Compensating Transactions

Compensating Transactions are designed to undo the effect of the original transactions that have aborted. The compensating transactions are triggered and started when the original transactions fail. Otherwise, the compensating transactions are not initiated. Once a compensating transaction has started, it must commit. In other words, the compensating transactions cannot abort. If a compensating transaction fails, it will be restarted. Compensating transactions are helpful to get back to the previous consistent state of the database. Compensating transactions do not ensure the entire restoration of the database initial state.

Example: Consider a scenario in a travel booking system that involves multiple services such as flight booking, hotel reservation, and car rental. Each of these services is handled by different systems and may involve multiple steps that take a significant amount of time. After the flight is booked and the hotel is reserved, an error occurs while renting the car. Since the entire transaction cannot be rolled back automatically due to the distributed nature of the systems and their respective commit points, compensating transactions are used to undo the effect of original transaction.

b) Weak level of Consistency

It allows distinct views of the database state to see different and unmatched updates in the database state. Causal Consistency and Eventual Consistency are weaker levels of Consistency.

- **Casual Consistency:** Causal Consistency requires only related operations to have a global ordering between them.

W: A = 1					
	R: A = 1	W: B = 2			
			R: A = 1	R: B = 0	R: B = 2
			R: B = 2	R: A = 1	

- **Eventual Consistency:** It is the weakest level of consistency. If there are no writes for a long period of time, the threads will eventually agree on the value of the last

write.

W: A = 1					
	R: A = 1	W: B = 2			
			R: A = 1	R: B = 0	R: B = 2
			R: B = 2	R: A = 0	

Write conditions of 2NF and 3NF. Suggest a set of functional dependencies so that relation schema R (A, B, C, D) with candidate key as {AB} satisfies 2NF but not 3NF conditions.

Second Normal Form (2NF) Conditions:

- The relation must be in 1NF.
- No non-prime attribute (i.e., not part of any candidate key) should be functionally dependent on a proper subset of any candidate key.

Third Normal Form (3NF) Conditions:

- The relation must be in 2NF.
- No non-prime attribute should be transitively dependent on any candidate key.

Consider the following set of functional dependencies:

$AB \rightarrow C$

A→D

These functional dependencies ensure that the relation is in 2NF because:

- The relation is already in 1NF.
- AB is a candidate key, and C and D are functionally dependent on the entire candidate key (AB).

However, the relation is not in 3NF because:

- A→D introduces a transitive dependency: A determines D, and D determines C (since D is not a prime attribute and D→C). This violates the 3NF condition.

Differentiate between star schema and snow flake schema representation of data warehouse.

Star Schema	Snowflake Schema
Contains fact tables and the dimension tables.	Does not contain fact tables, dimension tables as well as sub dimension tables.

Star Schema	Snowflake Schema
Top-down model.	Bottom-up model.
Uses more space.	Uses less space.
Takes less time for the execution of queries.	Takes more time for the execution of queries.
Normalization is not used.	Both normalization and denormalization are used.
Simple design	Complex design
Lower query complexity.	Higher query complexity
Very simple to understand	Difficult to understand.
Less number of foreign keys.	More number of foreign keys.
Higher data redundancy.	Lower data redundancy.

Explain Multivalued dependency with example.

Multivalued Dependency(MVD) means that for a single value of attribute 'a' multiple values of attribute 'b' exist. We write it as,

```
a --> --> b
```

It is read as a is multi-valued dependent on b.

Conditions for MVD

Any attribute say **a** multiple define another attribute b; if any legal relation r(R), for all pairs of tuples t1 and t2 in r, such that,

```
t1[a] = t2[a]
```

Then there exists t3 and t4 in r such that.

```
t1[a] = t2[a] = t3[a] = t4[a]
t1[b] = t3[b]; t2[b] = t4[b]
t1 = t4; t2 = t3
```

Example:

	a	b	c
	NAME	PROJECT	HOBBY
t1	Geeks	MS	Reading
t2	Geeks	Oracle	Music
t3	Geeks	MS	Music
t4	Geeks	Oracle	Reading

Condition-1 for MVD

t1[a] = t2[a] = t3[a] = t4[a]

Finding from table,

t1[a] = t2[a] = t3[a] = t4[a] = Geeks

So, condition 1 is Satisfied.

Condition-2 for MVD

t1[b] = t3[b]
And
t2[b] = t4[b]

Finding from table,

t1[b] = t3[b] = MS
And
t2[b] = t4[b] = Oracle

So, condition 2 is Satisfied.

Condition-3 for MVD

∃c ∈ R-(a ∪ b) where R is the set of attributes in the relational table.
t1 = t4
And
t2=t3

Finding from table,

t1 = t4 = Reading
And
t2 = t3 = Music

So, condition 3 is Satisfied. All conditions are satisfied, therefore,

a --> --> b

According to table we have got,

name --> --> project

And for,

a --> --> C

We get,

name --> --> hobby

Hence, we know that MVD exists in the above table and it can be stated by,

name --> --> project

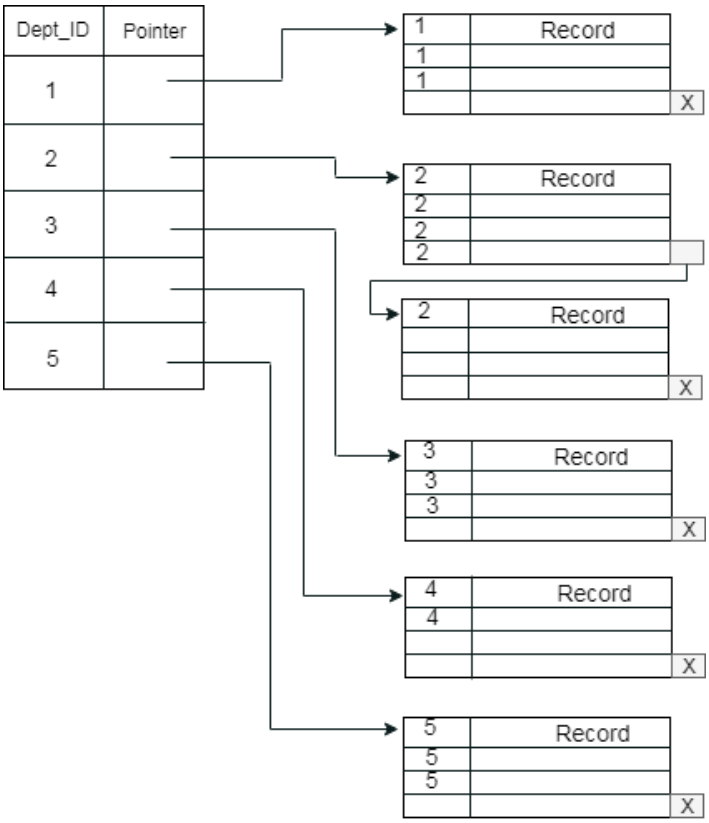
name --> --> hobby

Secondary and Clustering file index

Clustering Index:

- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

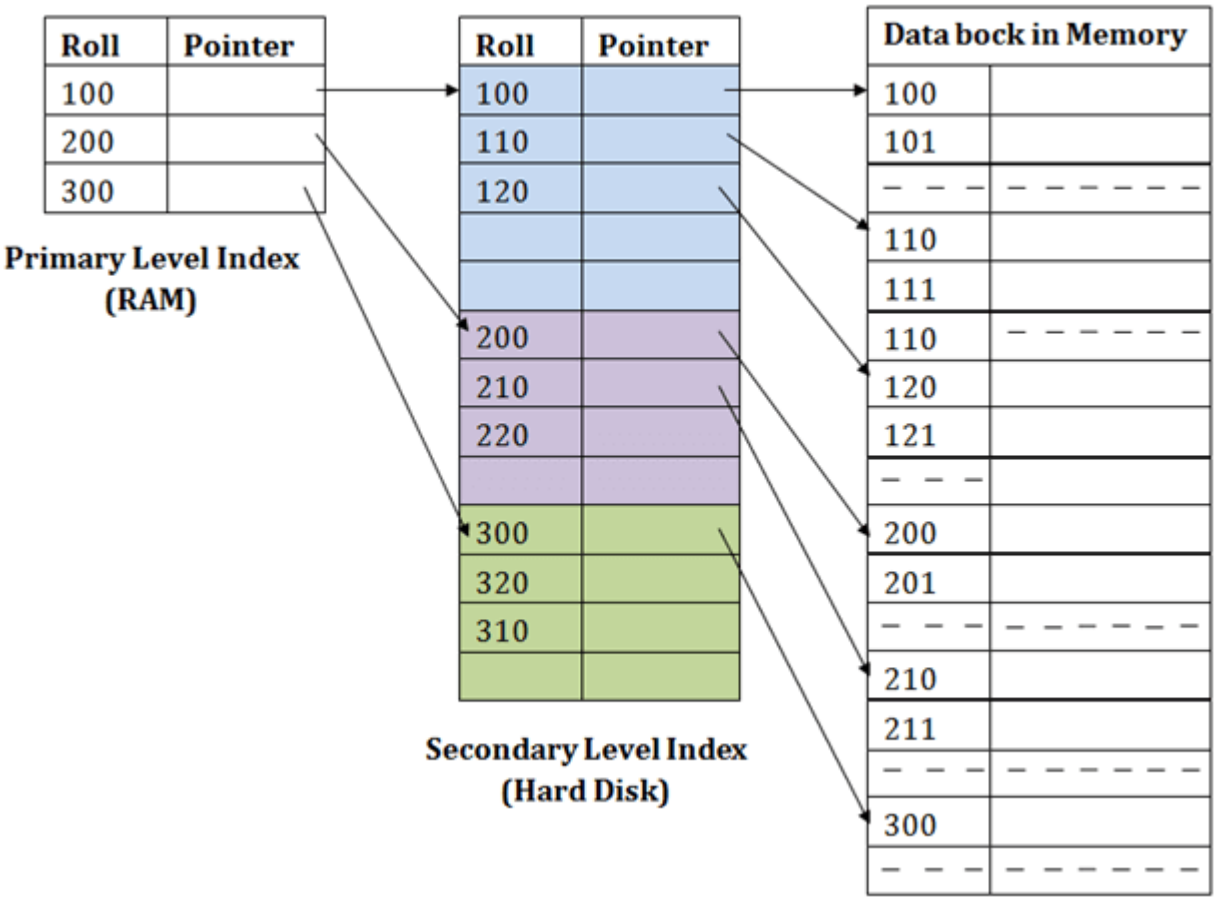
Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.



Secondary Index:

In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The

mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).



For example:

- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.