DATABASE

It is a collection of **inter – related data** in a **well-organized manner** which represents an aspect of the **real world.** The DB allows for retrieval, continuous modification and data search operations. Prior to the DB, a **file system** was used which stored permanent records aka files. There were a few drawbacks wit this system –

- Data redundancy
- Inconsistent data
- Maladaptive data
- Insecure data
- Incorrect data

The DB resolved these issues. We have a few key terms related to DBMS –

- **DB Schema –** It is the skeletal design of the database.
- **Data Constraints –** These are the restrictions that are applied on the data.
- **Data Dictionary/Metadata –** It is the data of the data. It is the combination of Schema and Constraints.
- **DB Instance –** It is used to define DB environment and its components.
- **Query –** It is a way to retrieve the data from the DB
- **Data manipulation –** We can either Insert, Update or Delete data from DB
- **Data Engine –** It is a component needed to create and manage DB queries.

ADV OF DB

1.Reduces database data redundancy to a great extent

2.The database can control data inconsistency to a great extent

3.The database facilitates sharing of data.

4.Database enforce standards.

5.The database can ensure data security.

6.Integrity can be maintained through databases.

DISADV OF DB

1.Security may be compromised without good controls.

2.Integrity may be compromised without good controls.

3.Extra hardware may be required

4.Performance overhead may be significant.

5.The system is likely to be complex.

DB TYPES

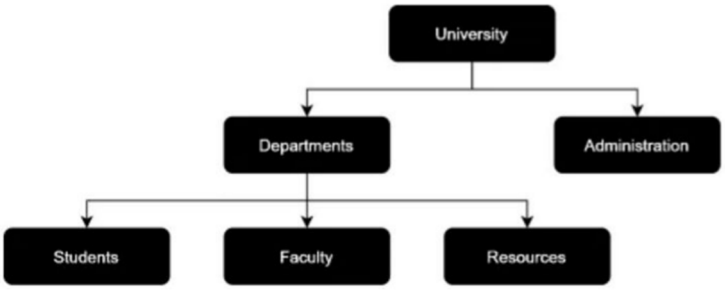
DB can be of mainly four types –

- Hierarchical

- Network
- Object – Oriented
- Relational

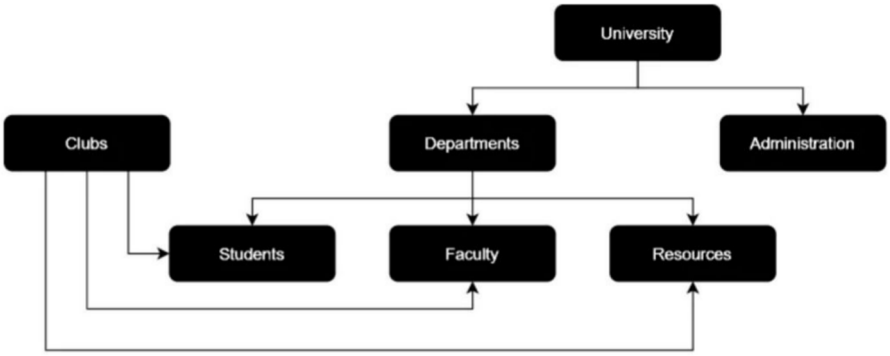
HIERARCHICAL DB

The DB is arranged in levels where each parent can have multiple children but **each child has a single parent.**



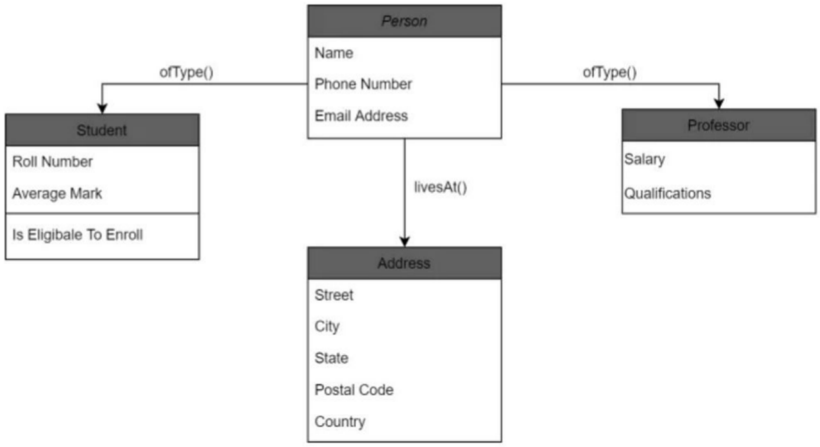
NETWORK DB

It is the hierarchical DB except here a child can have multiple parents.



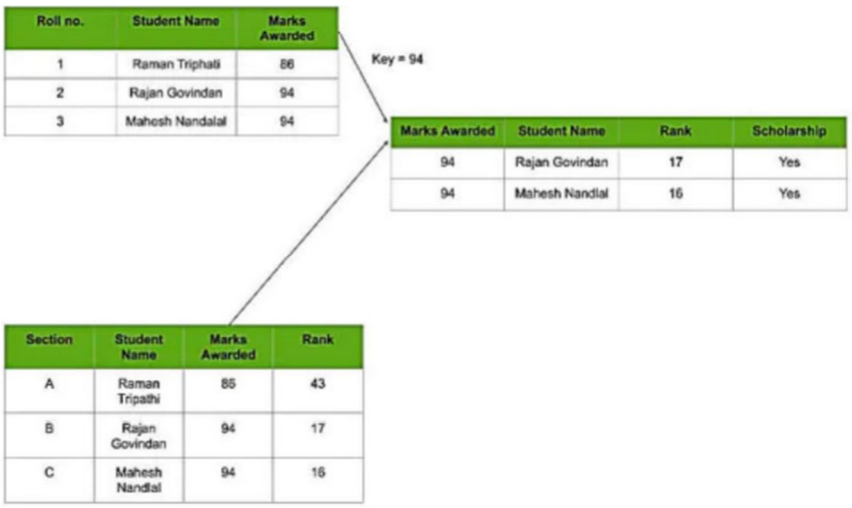
OBJECT – ORIENTED DB

The DB is arranged as classes and objected and they are connected to each other via methods.



RELATIONAL DB

This is the most used DB. It has information stored in tables as key – value pairs where every piece of info is related to every other info. Every DB has a primary key to uniquely identify the records.



DATABASE MANAGEMENT SYSTEM (DBMS)

DBMS is the software aka group of programs which handle the data in the DB. DBMS acts as an **interface between user and OS.** It takes requests from user and gives it to the OS for servicing the requests. There can be 3 types of users in a DBMS –

- **Application Programmers –** These users write the programs to interact with the data
- **DB Admin –** They manage the entire DBMS and DB environment
- **End users –** They are the ones that send the requests to manipulate the data.

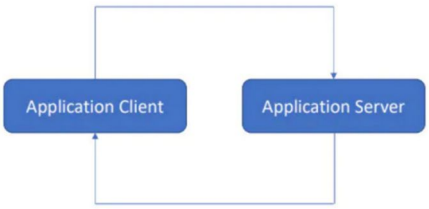
DBMS	File system
Collection of data.	Collection of data.
The user is not required to write the procedures.	The user has to write the procedures for managing the DB.
Gives an abstract view of data that hides the details.	Provides the detail of the data representation and storage.
Provides a crash recovery mechanism. (Protects the user from the system failure)	Doesn't have a crash mechanism. (If the system crashes while entering some data, then the content of the file will lost)
Provides a good protection mechanism.	Very difficult to protect a file under the file system.
Contains a wide variety of sophisticated techniques to store and retrieve the data.	Can't efficiently store and retrieve the data.
Takes care of concurrent access of data using locking.	Concurrent access has many problems like redirecting the file while other deleting or updating.

As we can see, DBMS has a lot of advantages. However, it is also very expensive to implement and at the same time can increase the complexity of the system.

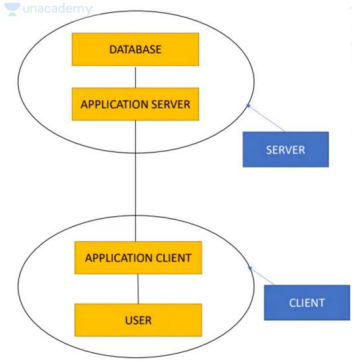
TYPES OF DBMS ARCHITECTURE

DBMS architecture can be of 3 types –

- **Single tier architecture –** In this case, the DB is available on the client system itself and thus there is no need for any network connection or service – request model to manipulate the data.
- **Two tier architecture –** In this case, the client sends a request to the server and then receives a response. The maintenance and understanding is better here but the system loses performance for large number of users.
- **Three tier architecture –** In this case, the client doesn't directly interact with the server. Rather, it interacts with the application server which in turn interacts with the DB to execute the query and send the response back. This is used when we have large web applications.



Two tier architecture

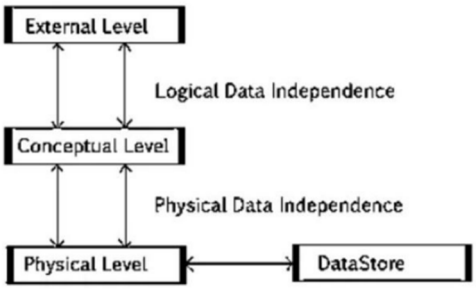


Three tier architecture

DBMS SCHEMA TYPES

- **Physical/Internal Schema –** How the data is stored in blocks
- **Logical/Conceptual Schema –** How data records are logically stored in the form of tables. DBA and programmers work on this level
- **View/Internal Schema –** Manages the DB views. This is where end users interact with the DBMS.

These insights has helped the DB engineers to divided the DBMS into three level independent modules –



DATABASE ABSTRACTION

Abstraction is the process of hiding unnecessary data and only displaying the required data.

Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.
Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.
View level: Highest level of data abstraction. This level describes the user interaction with database system.

DATA INDEPENDENCE

It is the property where a change in one data level should not affect the other data level. This can be of two types –

- **Physical –** Change in physical location should not change the conceptual level or external view of the data.
- **Conceptual –** Change in conceptual data should not affect the data at the external view level.

FEATURES OF DBMS

- A DBMS has tools to create and modify data models
- DBMS has queries for storing and retrieving data
- DBMS has mechanisms to control the situation when multiple users access the DB aka **concurrency control.**
- DBMS provides data security and integrity.
- DBMS also has backup mechanisms for easy data recovery.

7.RDBMS: Data is organized in the form of tables and each table has a set of rows and columns. The data is related to each other through primary and foreign keys.
8.NoSQL: Data is organized in the form of key-value pairs, document, graph, or column-based. These are designed to handle large-scale, high-performance scenarios.

DATA LANGUAGES

These are of 4 types –

- **Data Definition Language (DDL) –** Deals with DB schemas and descriptions of how data resides in the database. For eg – CREATE, ALTER, DROP, TRUNCATE, COMMENT, RENAME.
- **Data Manipulation Language (DML) –** Deals with data manipulation and change. For eg – SELECT, INSERT, DELETE, UPDATE, MERGE, CALL, LOCK TABLE, EXPLAIN PLAN
- **Data Control Language (DCL) –** Deals with the granting and revoking permissions using the GRANT and REVOKE query.
- **Transaction Control Language (TCL) –** Manages the transactions using ROLL BACK, COMMIT and SAVE POINT.

•CREATE: to create a database and its objects like (table, index, views, store procedure, function, and triggers)
•ALTER: alters the structure of the existing database
•DROP: delete objects from the database
•TRUNCATE: remove all records from a table, including all spaces allocated for the records are removed
•COMMENT: add comments to the data dictionary
•RENAME: rename an object

•SELECT: retrieve data from a database

•INSERT: insert data into a table

•UPDATE: updates existing data within a table

•DELETE: Delete all records from a database table

•MERGE: UPSERT operation (insert or update)

•CALL: call a PL/SQL or Java subprogram

•EXPLAIN PLAN: interpretation of the data access path

•LOCK TABLE: concurrency Control

•GRANT: grant permissions to the user for running DML(SELECT, INSERT, DELETE,...) commands on the table

•REVOKE: revoke permissions to the user for running DML(SELECT, INSERT, DELETE,...) command on the specified table

•Roll Back: Used to cancel or Undo changes made in the database

•Commit: It is used to apply or save changes in the database

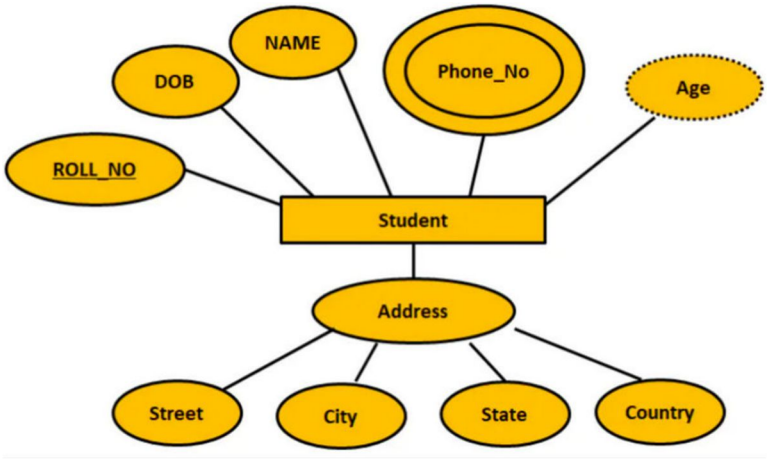
•Save Point: It is used to save the data on the temporary basis in the database

ENTITY RELATIONSHIP MODEL (ER MODEL)

This is a conceptual model that gives the graphical representation of the logical structure of the DB. It shows how the different entities are related (hence the name). Here are some standard definitions –

- **Entity Set –** Set of all entities.
- **Attributes –** The properties that define an entity. They are represented in an oval shape
    - **Key Attribute –** The attribute that can be used to uniquely identify each entity
    - **Composite Attribute –** The attribute that is composed of multiple attributes
    - **Multivalued Attribute –** The attribute that has more than one value for a given entity. It is represented by a **double oval.**
    - **Derived Attribute –** An attribute that can be derived from other attributes of the entity type. It is represented by a **dotted oval.**

For example,



Here,

- **Student** is the entity
- **ROLL_NO** is a **key attribute** which can be used to uniquely identify the student data.
- **Address** is a **composite attribute** which is formed by combining Street, City, State, Country.
- **Phone_No** is a **multivalued attribute** since a student can have multiple phone numbers.
- **Age** is a **derived attribute** which can be derived if we know the value of DOB attribute.

1. Strong Entity Set-

•A strong entity set possess its own primary key.
•It is represented using a single rectangle.

2. Weak Entity Set-

•A weak entity set do not possess its own primary key.
•It is represented using a double rectangle.

RELATIONSHIP

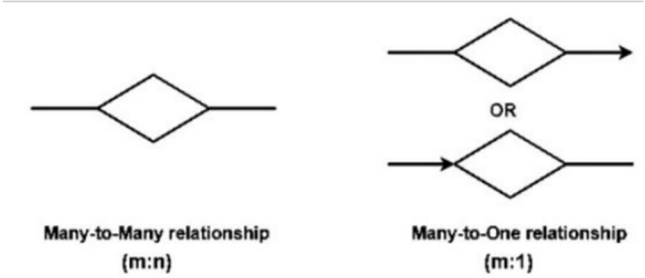
It defines the relationship between 2 entities and is represented by a **diamond shape.** For example, if a student is enrolled in a course, we can represent it as –

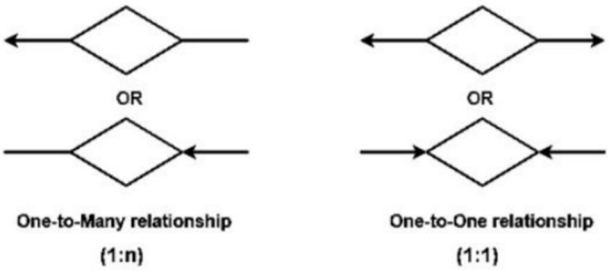
The number of entities participating in the relationship is called the **degree of relationship.**

- **Unary relationship –** 1 entity in the relationship
- **Binary relationship –** 2 entities in the relationship
- $\boldsymbol{n-ary}$ **relationship –** n entities in the relationship

Additionally, the number of times an entity can participate in a relationship will define the **cardinality of the relationship.**

- **One-to-one –** Marriage is a great example. For one groom, there is only one bride.
- **Many-to-one** – A parent child relationship. For multiple children, there can be a single parent.
- **Many-to-many** – Course and students. Multiple students can enroll in a single course and a single student can enroll in multiple courses.



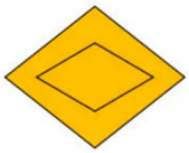


1. Strong Relationship Set-

•A strong relationship exists between two strong entity sets.
•It is represented using a diamond symbol.

2. Weak Relationship Set-

•A weak or identifying relationship exists between the strong and weak entity set.
•It is represented using a double diamond symbol.

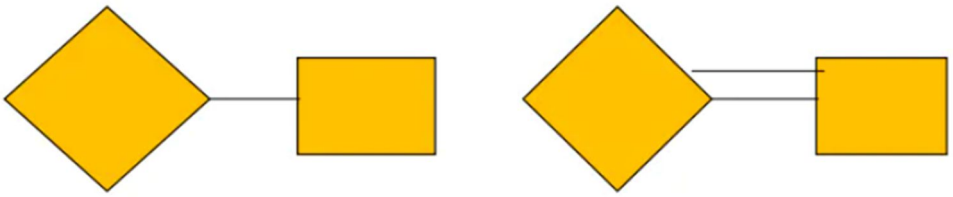


PARTICIPATION CONSTRAINTS

- **Total participation –** All entities in the set must take part in the relationship
- **Partial participation –** Not all entities are requiring to take part in the relationship

For example, let us take the relation of student and roll no. Each student must have a roll no and can have a unique roll no. Hence, it is a total participation with a one-to-one cardinality. On the other hand, when students enroll in a course, not all students require to enroll in the course. Hence, this is a partial participation relation which is also referred to as **Optional participation.**

•If minimum cardinality = 0, then it signifies partial participation.
•If minimum cardinality = 1, then it signifies total participation.



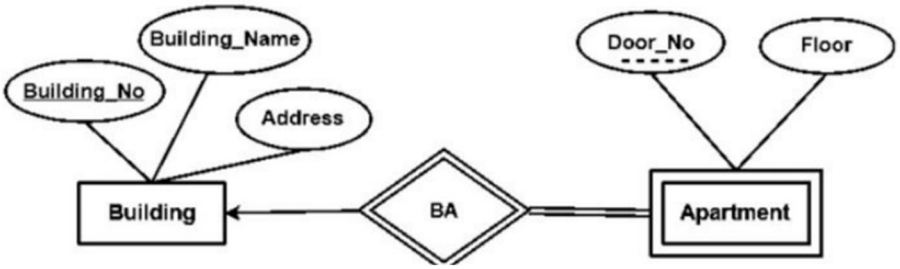
Partial participation *Total participation*

WEAK ENTITY SET

As mentioned previously, a weak entity set is one which doesn't have a primary key. On the other hand, a weak entity set has a **discriminator** which is a **partial key** that is formed by taking multiple attributes at a time. For example, let us assume that we have 2 sets – Building and Apartment. Building is a strong set as the building numbers are unique. On the other hand, multiple buildings can have the same apartment numbers making Apartment a weak set. Here, Apartment no. is a discriminator and is represented by a **dashed line.**

•A double rectangle is used for representing a weak entity set.
•A double diamond symbol is used for representing the relationship that exists between the strong and weak entity sets and this relationship is known as **identifying relationship**.
•A double line is used for representing the connection of the weak entity set with the relationship set.
•Total participation always exists in the identifying relationship.

Using the above symbols, we can represent the apartment and building relationship as follows –



NOTE

To form a relationship, we need a primary key. Hence, it is not possible to have a relationship between 2 weak sets.

RECURSIVE RELATIONSHIPS

It is the relationship defined between 2 entities of the same entity set. For example, let us take the example of an entity set of employees and a relationship called **REPORTS_TO.** Here,

- Any employee can report to any other employee who is senior to them
- The CEO doesn't report to anyone
- The junior most employee doesn't have anyone reporting to them

In this example, the relationship is recursive since we are defining it over the same set of Employee entities. At the same time, it is a **partial participation** as CEO doesn't report to anyone. In this relationship, suppose employee E1 reports to employee E2. Then, we can assign **role names** as Subordinate (E1) and Supervisor (E2).

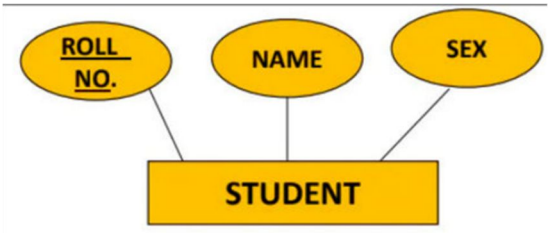
- Since CEO is not a subordinate to anyone, the minimum cardinality of Subordinate is 0
- Since every subordinate report to a single supervisor, the max cardinality of Subordinate is 1
- As the lowest level of employee can't be a supervisor, the min cardinality if Supervisor is 0
- We can have a case where all employees report to the CEO directly. In such a case, the max cardinality of the Supervisor will be N.

CONVERTING ER MODEL TO TABLES

One of the most important applications of ER model is to convert the same to the table. To do so, we need to follow a certain number of rules –

Rule 1

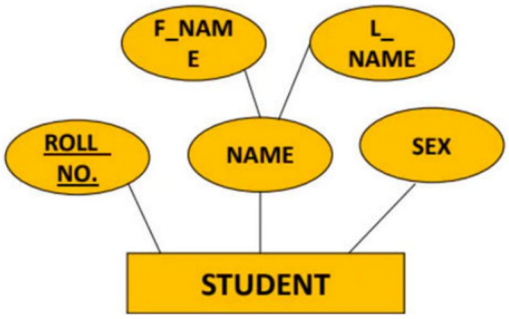
Any strong entity set with basic attributes can be represented using **1 table.**



ROLL_NO.	NAME	SEX

Rule 2

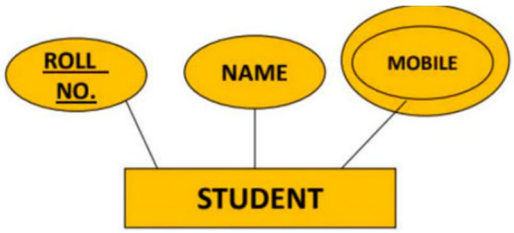
Suppose in the strong entity set, we have a **composite attribute.** In such a case, we still need only **1 table** but instead of mentioning the composite attribute, we create columns based on the components of the composite attribute.



ROLL_NO.	F_NAME	L_NAME	SEX

Rule 3

In the case of a strong entity set which has a **multivalued attribute,** we need to create a separate table for each multivalued attribute. Hence, if the number of multivalued attributes is $\boldsymbol{n}$**,** then we would need a total of $\boldsymbol{n+1}$ tables to represent the ER model.

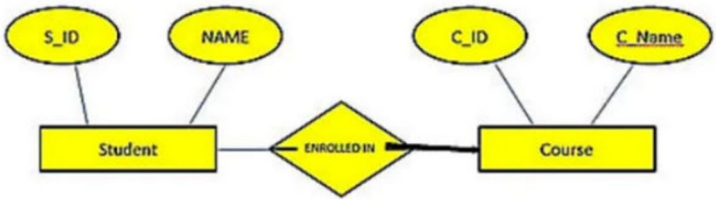


ROLL_NO.	NAME

ROLL_NO.	MOBILE

Rule 4

Just like the entity set, a relationship can also be converted to a **table** that includes the Primary key of the entity sets in the relationship.



In this case, the relationship **ENROLLED_IN** can be represented as –

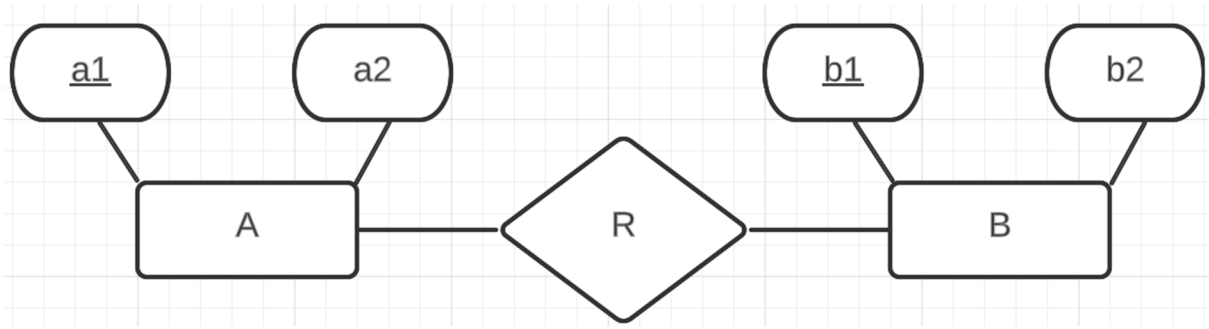
S_ID	C_ID

Rule 5

This rule deals with the binary relationships and the cardinality ratios of the same. Hence, there can be 4 total cases here –

- Binary relationship with many-to-many cardinality
- Binary relationship with many-to-one cardinality
- Binary relationship with one-to-many cardinality
- Binary relationship with one-to-one cardinality

Let us assume the binary relationship as follows –



$$A(a1, a2)$$

$$B(b1, b2)$$

Many-To-Many

In this case, we need 1 table for A, 1 for B and 1 for R. Total number of tables is **3.**

$$A(a1, a2)$$

$$B(b1, b2)$$

$$R(a1, b1)$$

Many-to-One

In this case, many entities of A are mapped to a single entity of B. For such cases, we can **combine A & R** in a single table. Thus, we need **2 tables.**

$$AR(a1, a2, b1)$$

$$B(b1, b2)$$

One-to-Many

Like the previous case, here we combine B & R into a single table thus causing us to **need 2 tables.**

$$A(a1, a2)$$

$$BR(b1, b2, a1)$$

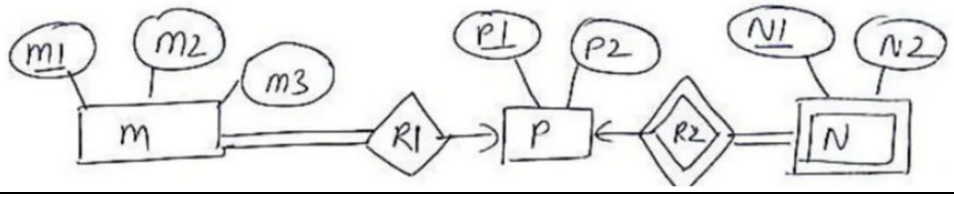
One-to-One

In this case, we have 3 cases –

- Both sides have partial participation, then we can combine either A&R or B&R
- Only one side has partial participation, then we can combine A, B and R as 1 table
- Both sides have total participation, then also we can combine A, B and R as 1 table

Question

Find the minimum number of tables needed to represent this ER model.



Answer

From the ER model, we can see that $M \rightarrow P$ is a many-to-one relationship and $N \rightarrow P$ is also a many-to-one relationship. Therefore, we need

$$MR1\left(\underline{m1}, m2, m3, \underline{p1}\right)$$

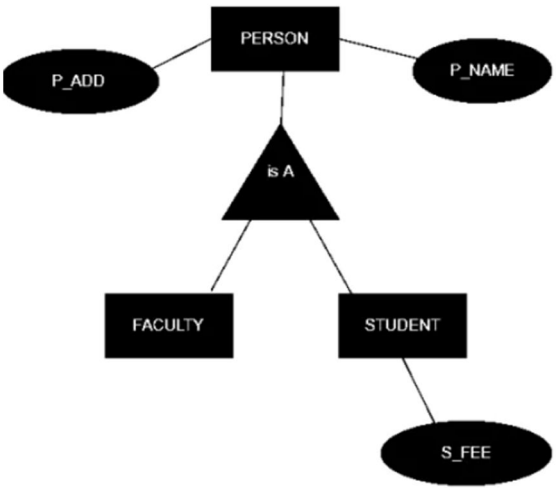
$$NR2\left(\underline{n1}, n2, \underline{p1}\right)$$

$$P\left(\underline{p1}, p2\right)$$

Thus, we need **3 tables.**

GENERALIZATION

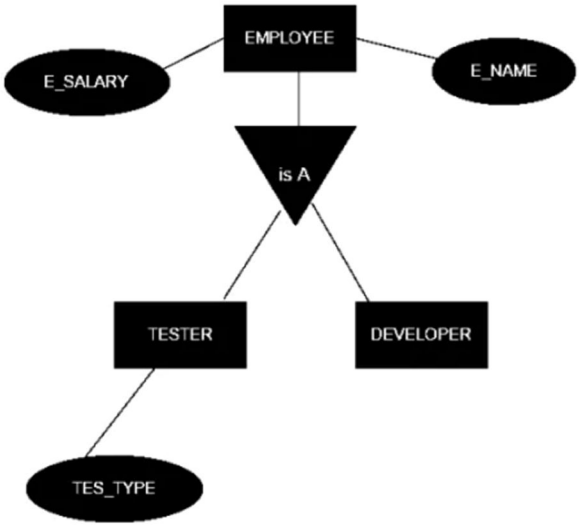
This is a **bottom – up approach** where we take multiple entities which have **something in common** and generalize it to a higher-level entity. For example, the ER model is shown below –



In this case, entities FACULTY and STUDENT are generalized into a higher – level entity called PERSON. Here, P_NAME and P_ADD are the **common attributes** since both Faculties and Students have name and address. However, S_FEE is a **specialized attribute** since only a Student pays fees.

SPECIALIZATION

In this case, a higher – level entity is specialized into multiple lower – level entities. This is a **top – down approach.** For example,



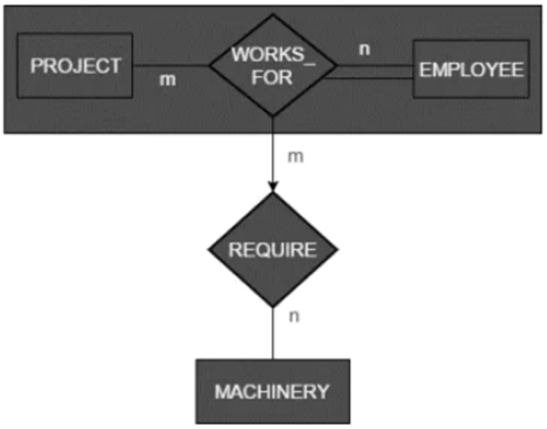
Here, the EMPLOYEE entity is specialized into either a TESTER or DEVELOPER. The Salary and Name are common attributes since they are both present for TESTER and DEVELOPER. However, TEST_TYPE is only for TESTER and hence is a specialized attribute.

GENERALIZATION	SPECIALIZATION
Generalization works in Bottom-Up approach.	Specialization works in top-down approach.
In Generalization, size of schema gets reduced.	In Specialization, size of schema gets increased.
Generalization is normally applied to group of entities.	We can apply Specialization to a single entity.

Generalization can be defined as a process of creating groupings from various entity sets	Specialization can be defined as process of creating subgrouping within an entity set
In Generalization process, what actually happens is that it takes the union of two or more lower-level entity sets to produce a higher-level entity sets.	Specialization is reverse of Generalization. Specialization is a process of taking a subset of a higher level entity set to form a lower-level entity set.
Generalization process starts with the number of entity sets and it creates high-level entity with the help of some common features.	Specialization process starts from a single entity set and it creates a different entity set by using some different features.
In Generalization, the difference and similarities between lower entities are ignored to form a higher entity.	In Specialization, a higher entity is split to form lower entities.
There is no inheritance in Generalization.	There is inheritance in Specialization.

AGGREGATION

Using regular ER models, we can define relationships between an entity and another entity. However, there can be cases where we need to define relationship between an entity and a relationship. In this case, we use **aggregation** where we combine a relationship between entities as a **high – level entity** and then that can be used to form a relationship further. For example,



In the above case, we are combining the WORKS_FOR relationship into a high – level entity. And then it is used to form the REQUIRE relationship. Hence, we can now form a relationship as a **Employee working on a project that requires a certain machinery.**

BINARY RELATIONS SUMMARY

CARDINALITY	PARTICIPATION	MIN. NO OF TABLES
Many-to-Many	Partial, both sides	3
	Partial, one side	2
	Total, both sides	1
Many-to-One	Partial, both sides	2
	Partial, one side	2
	Total, both sides	1

One-to-One	Partial, both sides	2
	Partial, one side	1
	Total, both sides	1

RELATIONAL MODEL

It is a way of defining how the data will be stored in RDBMS. The data is stored in RDBMS is in the form of tables and rows.

- Here, instead of entities we deal with **relations.**
- A property that defines a relation is called an **attribute** here.
- The name and attributes of a relation is called a **relation schema.** If there are more than 1 relation schema, then it is called **relational schema.**
- The set of tuples at a particular instance of time is called the **relation instance.**
- Number of attributes in a relation is termed as the **degree** of the relation
- The number of tuples in a relation is termed as **cardinality.**

RELATION INTEGRITY CONSTRAINTS

Relation constraints in general are restricts placed on the relations. Relation integrity constraints are set of rules to ensure quality of data and prevent accidental damage from DB. These can be of 4 types –

- **Domain Constraints –** Ensures that the value entered in the cell is part of the domain aka allowed values. For eg, a field called AGE can only house positive integers.
- **Entity Integrity constraints –** This states that the primary key can't be NULL. However, other values can be null.
- **Referential Integrity constraints –** This is present between tables. If the foreign key of table A refers to primary key of table B, then the foreign keys need to be either present in table B or should be NULL. Basically, Foreign key of A must be a subset of Primary key of B.
- **Key Constraints –** The Primary key must be unique

PREVENTION OF ANOMALY

An **anomaly** is basically an irregularity. Usually, these occur in 3 process – Insert, Delete and Update. The **Referential Integrity constraints** can be used to prevent these anomalies.

Suppose we have 2 tables as shown below –

ROLL_NO	NAME
1	ABC
2	DEF

TABLE A

EMP ID	ROLL_NO
10	1
20	2

TABLE B

In this case, ROLL_NO is the primary key in Table A (**referenced relation)** and a foreign key in Table B **(referencing relation)**. Now, let us assume there is an insertion in Table B as follows –

EMP ID	ROLL_NO
10	1
20	2
30	3

TABLE B

Post the insertion, the **foreign key is no longer a subset of the primary key.** This is called **Insert anomaly in referencing relation.** This causes Referential Integrity violation. To resolve this, one of the methods used is to deny the update request if it will cause a Referential Integrity violation.

Now, let us assume the tables as follows –

ROLL_NO	NAME
1	ABC
2	DEF

TABLE A

EMP ID	ROLL_NO
10	1
20	2

TABLE B

Now, let's say that we delete the row with ROLL_NO = 2 in Table A (referenced relation). In that case, the Foreign key will no longer be a subset of Primary key and hence this is a Referential Integrity violation. This is called the **Deletion anomaly in referenced relation.** To resolve this, we can employ 3 methods –

- If the value is deleted in Table A (referenced relation), then delete the tuple corresponding to the same foreign key in Table B as well (referencing relation). In our example, if we have deleted ROLL_NO = 2, then we need to delete tuple with EMP_ID = 20 from Table B as well. This is called **On Delete Cascade.**
- The second solution is to outright deny the delete requests that cause referential integrity violation
- Finally, we also just simply set the value of the foreign key value to NULL. For example, if we update the ROLL_NO = 2 to be ROLL_NO = 5 in referenced relation, then we can set the value of EMP_ID = 20 (ROLL_NO = 2) to NULL.

Finally, let's again take the tables as follows –

ROLL_NO	NAME
1	ABC
2	DEF

TABLE A

EMP ID	ROLL_NO
10	1
20	2

TABLE B

Now, let's assume we have gotten a case to update the referenced table (Table A) as follows –

ROLL_NO	NAME
1	ABC
5	DEF

TABLE A

Now, the foreign keys are no longer a subset of primary keys and hence there is a Referential Integrity violation. This is called **Update anomaly in referenced relation.** To solve this, we have again 3 solutions –

- If the primary key in referenced relation is updated, update the same foreign key as well in referencing relation. This is called **On Update Cascade.**
- Deny the update requests that cause the anomaly and integrity violations
- If the primary key is updated, then set the previous foreign key value to NULL

TYPES OF KEYS

- **Super Key –** It is a set of attributes that can be used to uniquely identify each tuple.
- **Candidate Key –** The minimal super key is called a candidate key. The candidate keys have to be unique and NOT NULL. At the same time, the attributes used in the candidate keys are called **prime attributes.** A relation can have multiple candidate keys.
- **Primary Key –** Primary key is a candidate key that the designer uses to uniquely identify the tuples in a relation. There can only be **one primary key in a relation.** $PK \subseteq CK \subset SK$
- **Alternate Key –** The candidate keys that are not primary keys are called alternate keys.
- **Foreign Key –** A key X is said to be a foreign key if it is present in the referencing relation and such that key X is dependent on attribute key Y in the referenced relation. Foreign key can be null and need not be unique. It is a subset of primary key. $X \subset Y$
- **Composite Key –** It is a key made from multiple attributes. It is also referred to as a **compound key.**
- **Partial Key –** It is a key which can't be used to uniquely identify the tuples
- **Secondary Key –** It is used for indexing and improving the speed of data access.
- **Surrogate Key –** It is a key that is unique for all tuples, can be updated and can't be NULL.
- **Unique Key –** It is a key that is unique for all tuples and can't be updated. However, it can be NULL.

KEYS	Unique for all records	Can be updated?	Can be NULL?
Primary	Yes	No	No
Surrogate	Yes	Yes	No
Unique	Yes	No	Yes

Question

Consider the relation as shown below –

Student ID	Student Name	Student Email	Student Age	CPI
2345	Shankar	Shankar@math	X	9.4
1287	Swati	swati@ee	19	9.5
7853	Shankar	shankar@cse	19	9.4
9876	Swati	swati@mech	18	9.3
8765	Ganesh	ganesh@civil	19	8.7

If (Student Name, Student Age) is to be a key in this instance, then what values can X take?

Answer

In the relation, we have 2 records –

- (Shankar, 19)
- (Shankar, X)

Since we need the key to be unique, $\boldsymbol{X \neq 19}$

Question

Which of the following is NOT a superkey in a relational schema with attributes V, W, X, Y, Z and primary key V Y ?
(A) V X Y Z
(B) V W X Z
(C) V W X Y
(D) V W X Y Z

GATE 2016

Answer

Option B

Question

Let R = (A, B, C, D, E, F) be a relation scheme with the following dependencies-
C → F
E → A
EC → D
A → B

Which of the following is a key for R?
1.CD
2.EC
3.AE
4.AC

Also, determine the total number of candidate keys and super keys.

Answer

A candidate key will be the one which can derive every other key. In this question,

$$EC \rightarrow D$$

But, we know that $E \rightarrow A$ and $C \rightarrow F$. So,

$$EC \rightarrow DAF$$

Now, we know that $A \rightarrow B$. Additionally, any key can derive itself so $EC \rightarrow EC$. Therefore, we get –

$$EC \rightarrow DAFBEC$$

Hence, EC is the **candidate key.** Now, to get the number of super keys, we need to form combinations of EC with the other keys. Thus,

$$E\ C\ \_\_\_\_$$

We have a slot of each of the other attributes. Now, these attributes can participate or not. Hence,

$$Total\ no\ of\ super\ keys = 2^4 = \mathbf{16}$$

NOTE

In general, we have –

$$No\ of\ super\ keys = 2^{(No\ of\ attributes - No\ of\ attributes\ in\ Candidate\ key)}$$

NORMALIZATION

This is the process of organizing the data in DB which minimizes the redundancy between the relations. It also helps in resolving/eliminating insert, update and delete anomaly.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form (1NF)

In this form, no cell has **composite values.** There are only **atomic values.** No two rows are identical and every row has unique values.

Second Normal Form (2NF)

In this case, first the table has to be in 1NF form. In addition to that, there should be no **partial dependency.** This means that no **non-prime attribute (NPA)** should be dependent on a **partial candidate key.** For example, let's take the case as follows –

$$R(A, B, C, D, E, F)$$

$$BC \rightarrow ADEF$$

In this case, the prime attributes are B, C and the NPAs are A, D, E, F. We can see that all NPAs are dependent on the entire candidate key. Hence, it **has no partial dependency.** Suppose, we now add another dependency –

$$B \rightarrow F$$

In this case, a NPA F is dependent on B which is a partial candidate key (aka part of the candidate key). Hence, there is **partial dependency** and hence the table will **NOT be in 2NF.**

Third Normal Form (3NF)

In this case, the table first has to be in 2NF form. In addition to that, it must **not have transitive dependency.** This means that there shouldn't be any NPA deriving another NPA. For example,

$$R(A, B, C, D, E, F)$$

$$BC \rightarrow ADEF$$

This is the relation in 2NF form. Here, there is no NPA deriving another NPA. Hence, this table is in 3NF form. Suppose, we add the following dependency –

$$D \rightarrow F$$

In this case, D which is a NPA is deriving F which is a NPA. Hence, this relation is not in 3NF form.

Boyce – Codd Normal Form (BCNF)

In this case, the relation must first be in 3NF form. In addition to that, for every derivation $X \rightarrow Y$, $\boldsymbol{X}$ **must be a super key**. For example,

$$R(A, B, C, D, E, F)$$

$$BC \rightarrow ADEF$$

In this case, BC is the candidate key and hence every super key must have BC in it. In the above relation, the relation is in BCNF as BC is a super key. However, let's say we have the following derivation –

$$D \rightarrow B$$

In the above case, D does not contain BC and hence is **not a super key.** Therefore, the relation is no longer in BCNF form.

Fourth Normal Form (4NF)

In this case, the relation must already be in BCNF form. In addition to that, there **must not be any multivalued dependency.** For a dependency $A \rightarrow B$, is for a single value of A multiple values of B exist, then it is termed as multivalued dependency. For example,

PERSON	MOBILE	FOOD_LIKES
Mahesh	9893/9424	Burger / pizza
Ramesh	9191	Pizza

$$Person -\!\gg Mobile$$

$$Person -\!\gg Food\_Likes$$

Fifth Normal Form (5NF)

The relation must be in 4NF and must not have any join dependency. A join dependency means that the relation can be decomposed into lossless sub – relations. For a 5NF form, the decomposition must be lossy. The difference between lossy and lossless decomposition is explained later on. This is also called **Project – join Normal Form (PJ/NF)**

Question

Consider the following relation –

$$R(A, B, C)$$

$$AB \rightarrow C$$

$$B \rightarrow C$$

Find the normal form of this relation

Answer

By default, we first assume that the relation is in 1NF. Now, we can see that –

$$Candidate\ key = \{AB\} \rightarrow \{A, B, C\}$$

Hence, C is a NPA. Now, from the question we can see that NPA (C) is being derived/is dependent on partial candidate key (B). Thus, there is partial dependency and hence the relation is NOT in 2NF form. As it is not in 2NF form, it can't be in any other form as well. Hence, the relation is in **1NF form.**

Question

Which of the following is TRUE?
(A) Every relation in 3NF is also in BCNF
(B) A relation R is in 3NF if every non-prime attribute of R is fully functionally dependent on every key of R
(C) Every relation in BCNF is also in 3NF
(D) No relation can be in both BCNF and 3NF

Answer

Option C is the only true statement

Question

Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values.
The relation R is
F = {CH -> G, A -> BC, B -> CFH, E -> A, F -> EG}
(A) in 1NF, but not in 2NF. (B) in 2NF, but not in 3NF.
(C) in 3NF, but not in BCNF. (D) in BCNF

Answer

$$Candidate\ key = \{BD\} \rightarrow \{B, D, C, F, H, G, E, A\}$$

Thus, $\{A, C, E, F, G, H\}$ are NPAs. We can see that NPAs (CFH) are dependent on partial candidate key (B). Therefore, there is partial dependency and the relation is **NOT in 2NF.** Hence, **Option (a)** is correct.

NOTE

In relations where all attributes are prime attributes, the relation will always be **3NF** but **needn't be BCNF.** If there is a relation that is binary aka has only 2 attributes, then it is in **BCNF form.** BCNF form has **no redundancy.**

Trivial derivations are those derivations $X \rightarrow Y$ where $Y \subset X$. In case all derivations in a relation are trivial, the relation is **definitely BCNF.**

2NF is the normal form that is based on the concept of "*Full Functional Dependency*"

DECOMPOSITION

When we have a relation that is not in the required normal form, we **decompose** the relation into sub – relations. While decomposing into sub – relations, we need to ensure –

- No data is lost
- No dependency is changed

For example, let us assume we have a relation R that is decomposed into $R1$ and $R2$. Then, if $R1 \bowtie R2 \supset R$ (natural join of R1 and R2 is a superset of R), then the decomposition is **lossy.** On the other hand, if $R1 \bowtie R2 = R$, then the decomposition is **lossless.**

If a relation decomposition is lossless, then it needs to satisfy the following conditions –

- The union of the attributes of the sub – relations must be equal to the attributes of the original relation.

$$Att(R1) \cup Att(R2) = Att(R)$$

- There must be some common attributes between the sub – relations.

$$Att(R1) \cap Att(R2) \neq \phi$$

- At least one of the common attributes must be a candidate key for either of the sub – relations.

For example, let us take –

$$R(A, B, C, D)$$

$$A \rightarrow BC$$

Now, we are decomposing the relation R into 2 sub – relations as follows –

$$R1(A, B, C) \qquad R2(A, D)$$

We can now check for lossless conditions –

$$Att(R1) \cup Att(R2) = \{A, B, C, D\} = Att(R)$$

$$Att(R1) \cap Att(R2) = \{A\} \neq \phi$$

For $R1$, we can see from the derivation $A \rightarrow BC$ that A is the candidate key. Since this is also present as a common attribute between the sub – relations, it means that this **decomposition is lossless.**

One thing to note here is that since the derivation of R is also present in $R1$, the derivation is preserved and hence the **decomposition is also called dependency preserving.**

Question

Consider a schema R(MNPQ) and functional dependencies M → N, P → Q. Then the decomposition of R into R_1 (MN) and R_2(PQ) is_____.

UGC 2017

Answer

First, we check for lossless –

$$Att(R1) \cup Att(R2) = \{M, N, P, Q\} = Att(R)$$

$$Att(R1) \cap Att(R2) = \{\} = \phi$$

Hence, this decomposition is **lossy.** Now, we check for dependency preserving –

$$R1(M, N) \qquad M \rightarrow N$$

$$R1(P, Q) \qquad P \rightarrow Q$$

Hence, the dependencies of R are preserved by $R1$ and $R2$.

Therefore, the **decomposition is lossy and dependency preserving.**

FUNCTIONAL DEPENDENCY

A FD is a dependency $A \rightarrow B$ such that for a certain value of A, the value of B remains the same across instances.

Question

GATE Question: Consider the relation scheme R = {E, F, G, H, I, J, K, L, M, M} and the set of functional dependencies {{E, F} -> {G}, {F} -> {I, J}, {E, H} -> {K, L}, K -> {M}, L -> {N} on R. What is the key for R? (GATE-CS-2014)

A. {E, F}

B. {E, F, H}

C. {E, F, H, K, L}

D. {E}

Answer

Option B is the correct answer

Question

GATE Question: In a schema with attributes A, B, C, D and E following set of functional dependencies are given

{A -> B, A -> C, CD -> E, B -> D, E -> A}

Which of the following functional dependencies is NOT implied by the above set? (GATE IT 2005)

A. CD -> AC

B. BD -> CD

C. BC -> CD

D. AC -> BC

Answer

$$(CD)^+ = \{C, D, E, A, B\}$$

$$(BD)^+ = \{B, D\}$$

$$(BC)^+ = \{B, C, D, E, A\}$$

$$(AC)^+ = \{A, C, B, D, E\}$$

Hence, we can have

$$CD \rightarrow AC$$

$$BC \rightarrow CD$$

$$AC \rightarrow BC$$

Thus, **Options A,C and D** are correct.

TRIVIAL FD

A FD of the form $X \rightarrow Y$ is said to be trivial if $Y \subseteq X$. Suppose we have Y not a subset of X and $X \cap Y = \phi$, then we call this is a **completely non-trivial** dependency. If $X \cap Y \neq \phi$, then it is called a **semi – complete non-trivial** dependency.

INFERENCE RULES

These are rules that are based on Armstrong's axioms and are used to **derive additional FDs** from a given set of FDs. They are 7 inference rules –

- Reflexive Rule
- Augmentation Rule
- Transitive Rule
- Union Rule
- Decomposition Rule
- Pseudo Transitive Rule
- Composition

Reflexive Rule (IR1)

If $Y \subseteq X$, then we can derive a dependency $X \rightarrow Y$

Augmentation Rule (IR2)

If $X \rightarrow Y$, then we can derive $XZ \rightarrow YZ$

Transitive Rule (IR3)

If $A \rightarrow B$ and $B \rightarrow C$, then we can derive $A \rightarrow C$

Union Rule (IR4)

If $X \rightarrow Y$ and $X \rightarrow Z$, then we can derive $X \rightarrow YZ$

Decomposition Rule (IR5)

This is the reverse of the union rule. So, if $X \rightarrow YZ$, then we can derive $X \rightarrow Y$ and $X \rightarrow Z$

Pseudo Transitive Rule (IR6)

If $X \rightarrow Y$ and $YZ \rightarrow A$, then we can derive $XZ \rightarrow A$

Composition Rule (IR7)

If $X \rightarrow Y$ and $A \rightarrow B$, then we can derive $XA \rightarrow YB$

CANONICAL COVER

This is a simplified, reduced version of the set of FDs. It is an **irreducible set of FDs.** For a given FD set, the canonical cover needs to be lossless and at the same time it **need not be unique.** To get the canonical cover, we need to perform the following steps –

- First, write the FDs such that the RHS has a single element.
- Now, for every FD, find the closure.
- Next, ignore the FD and see if the closure remains the same or if it changes. If it remains the same, then we can ignore that FD

For example, let us consider the relation $R(W, X, Y, Z)$ with the following FDs –

$$X \rightarrow W$$

$$WZ \rightarrow XY$$

$$Y \rightarrow WXZ$$

As per Step 1, we decompose the FDs such that there is only 1 element in the RHS –

$$X \rightarrow W \qquad (1)$$

$$WZ \rightarrow X \qquad (2)$$

$$WZ \rightarrow Y \qquad (3)$$

$$Y \rightarrow W \qquad (4)$$

$$Y \rightarrow X \qquad (5)$$

$$Y \rightarrow Z \qquad (6)$$

Now, we take the FDs one by one and check the closure.

FD (1)

$$(X)^+ = \{X, W\}$$

Ignoring FD (1), we get –

$$(X)^+ = \{X\}$$

Since there is a difference, we **can't remove this FD.**

FD (2)

$$(WZ)^+ = \{W, Z, X, Y\}$$

Ignoring FD (2), we get –

$$(WZ)^+ = \{W, Z, X, Y\}$$

Therefore, we **can remove FD (2)**

We continue the same for all the FDs and finally, we get –

$$X \rightarrow W$$

$$WZ \rightarrow Y$$

$$Y \rightarrow X$$

$$Y \rightarrow Z$$

STRUCTURED QUERY LANGUAGE (SQL)

It is a language used for manipulating and accessing DBs. SQL is used in RDBMS where data is stored in the form of tables and records. Each column is called a **field.** There are some important SQL commands –

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

SQL Select Statement

Used to select data from the table. The general syntax is as follows –

```
SELECT column1, column2, ...
FROM table_name;
```

In case we want to select all the data from the table, we can use –

```
SELECT * FROM table_name;
```

To ensure that we get the **unique** values from the table, we can use the **DISTINCT** keyword –

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

SQL Where Statement

The WHERE clause is used to filter records.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL Order By Statement

The Order By clause is used to arrange the records in either ascending or descending order.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

In this case, first the records will be arranged as per *column1.* If there are certain rows which have the same value in *column1,* then the *column2* value is checked to check for order.

SQL Insert Statement

It is used to insert records in the table. The data can be inserted either by stating the columns or by just providing the values.

```
INSERT INTO TABLE_NAME
VALUES (value1, value2, value 3, .... Value N);

INSERT INTO TABLE_NAME
[(col1, col2, col3,.... col N)]
VALUES (value1, value2, value 3, .... Value N);
```

SQL Update Statement

It is used to update the data already present in the table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Suppose we remove the WHERE condition, then all the rows under that field will be updated.

SQL Delete Statement

It is used to delete rows from the table.

```
DELETE FROM table_name WHERE some_condi
tion;
```

If there is no where condition provided, then all the rows in the table will be deleted.

SQL Select Top

This statement is used to specify how many rows from the top that satisfy the conditions need to be returned.

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

We can also use percentages instead of count –

```
SELECT TOP 50 PERCENT * FROM Customers;
```

CustomerID	CustomerName	ContactName
1	Alfreds Futterkiste	Maria Anders
2	Ana Trujillo Emparedados y helados	Ana Trujillo
3	Antonio Moreno Taquería	Antonio Moreno

CustomerID	CustomerName	ContactName
1	Alfreds Futterkiste	Maria Anders
2	Ana Trujillo Emparedados y helados	Ana Trujillo

SQL Aggregate Functions

These are used to perform calculations on multiple rows of a single field. These functions include –

- **COUNT –** Used to get the number of rows. It includes duplicate data and can be applied to either numeric or non – numeric data.
- **SUM –** Used to get the sum of the row data. It works only for numeric data.
- **MIN –** Gives the min data
- **MAX –** Gives the max data
- **AVG –** Performs the average of the rows. It is basically SUM/COUNT

SQL Like Operator

This is used with the WHERE clause to check for a pattern.

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

In the Like operator, '%' operator checks for 0 or more occurrences. The '_' operator checks for a single character.

SQL RELATIONAL OPERATORS

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Question

Which of the following is/are correct?

(a) An SQL query automatically eliminates duplicates

(b) An SQL query will not work if there are no indexes on the relations

(c) SQL permits attribute names to be repeated in the same relation

(d) None of the above

[1999 : 2 Marks]

Answer

Option D is the correct answer. None of the statements are correct.

SQL LOGICAL OPERATORS

The AND and OR operators are used to filter records based on more than one condition:
•The AND operator displays a record if all the conditions separated by AND are TRUE.
•The OR operator displays a record if any of the conditions separated by OR is TRUE.
The NOT operator displays a record if the condition(s) is NOT TRUE.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3
...;
```

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3
...;
```

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

SQL JOINS

These statements are used to combine 2 or more table data. They can be of 4 types –

- **INNER –** This join will select the records in both the tables as long as the conditions are satisfied
- **FULL –** This will return the complete left and right table values as long as conditions are satisfied. If there are no matching values, it will return NULL.
- **LEFT -** This will return all the records of the left table and the corresponding matching values from the right table. If there are no matching values, then it will return NULL
- **RIGHT -** This will return all the records of the right table and the corresponding matched values from the left table. If there are no matched values, then it will return NULL

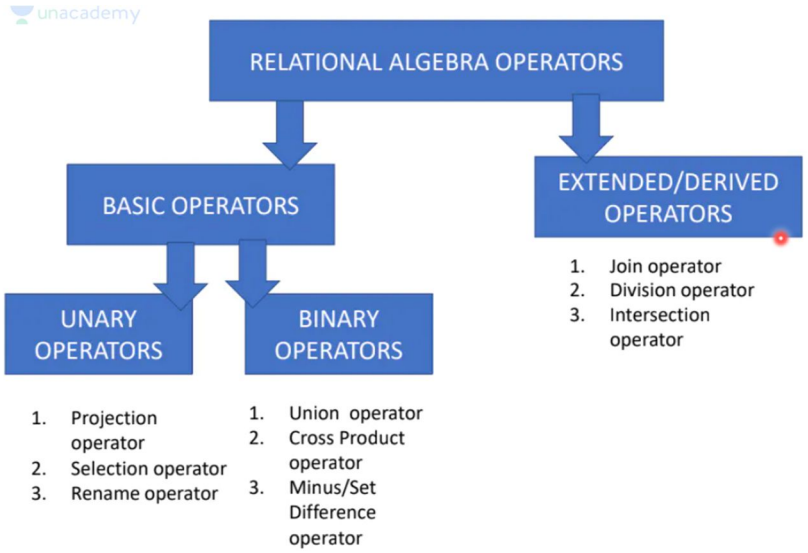
SQL Set Operation

It is used to join 2 SELECT statements. They can be of 4 types –

- **UNION –** This is used to combine the output of 2 SELECT statements. This also removes duplicate items.
- **UNIONALL –** It is the same as UNION except it doesn't remove the duplicate items.
- **INTERSECT –** This returns the intersection of the outputs of the 2 SELECT statements. It removes the duplicate items.
- **MINUS –** This returns the rows present in the first query but absent in the second.

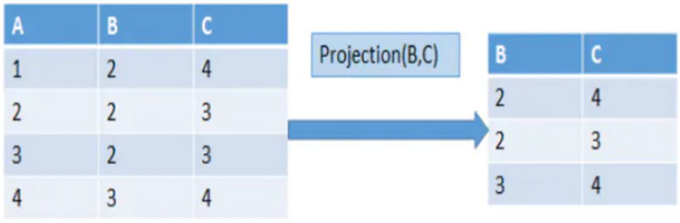
RELATIONAL ALGEBRA

It is a conceptual procedural query language which takes a relation as input and produces a relation as an output.



Projection (π)

Project required column data from a relation. By default, it removes duplicate data.



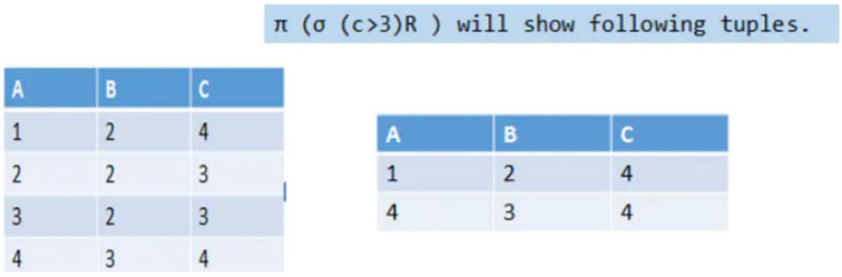
Projection is **NOT COMMUTATIVE.** Hence,

$$\pi_{<list2>}(\pi_{<list1>}(R)) \neq \pi_{<list1>}(\pi_{<list2>}(R))$$

Normally, the projection table's cardinality will be lesser than the original table. However, in case there are **no duplicate tuples,** the cardinality will be the **same.** For a relation of **degree = n,** we can have a total of $2^n - 1$ possible projections.

Selection (σ)

Selection is used to select the required tuples of the relations as per the condition. However, the selection doesn't display the data. For that, we need to use a projection.



Selection operator is **COMMUTATIVE.**

$$\sigma_{A \wedge B}(R) = \sigma_{B \wedge A}(R)$$

OR

$$\sigma_{B}(\sigma_{A}(R)) = \sigma_{A}(\sigma_{B}(R))$$

The **degree** of the input and selected relations **will be the same.** Minimum number of tuples selected will be 0 and the max number of tuples selected will be the number of tuples in R. Hence,

$$Cardinality \in [0, |R|]$$

Union ($\cup$)

This is the same as the set theory union. Except, the two tables which are being passed through Union must have the same set of attributes. Also, it removes **duplicate elements.** It is both commutative and associative.

Intersection $(\cap)$

This is the same as the set theory intersection. Except, the two tables which are being passed through Intersection must have the same set of attributes. Also, it removes the duplicate. It is both commutative and associative.

Difference $(-)$

This is the same as the set theory difference. Except, the two tables which are being passed through difference must have the same set of attributes. Also, it removes the duplicate. It is associative but not commutative.

Conditional Join $(\bowtie_c)$

Conditional Join is used when you want to join two of more relation based on some conditions. For example,

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

EMP_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
5	NARESH	HISAR	9782918192	22
6	SWETA	RANCHI	9852617621	21
4	SURESH	DELHI	9156768971	18

$$STUDENT \bowtie_{c\ STUDENT.ROLL\_NO>EMPLOYEE.EMP\_NO} EMPLOYEE$$

The command will join the Student and Employee tables and chooses the tuples that have a ROLL_NO greater than EMP_NO.

ROLL_NO	NAME	ADDRESS	PHONE	AGE	EMP_NO	NAME	ADDRESS	PHONE	AGE
2	RAMESH	GURGAON	9652431543	18	1	RAM	DELHI	9455123451	18
3	SUJIT	ROHTAK	9156253131	20	1	RAM	DELHI	9455123451	18
4	SURESH	DELHI	9156768971	18	1	RAM	DELHI	9455123451	18

One thing to note here is that **Join** is an **Extended/Derived operator.** This means that it can be derived from using the basic operators as well. This can be done as follows –

$$A \bowtie_{c\ (condition)} B = \sigma_{(condition)}(A \times B)$$

Where $A \times B$ is a **cartesian product** which will map each row of A with every row of B.

Division Operator $(\div)$

For the operation $A \div B$, the result will be the attributes that are in A but not in B aka $A - B$. This will only work when $B \subseteq A$. For example,

Student –

Student Name	Subject
Ashish	Machine Learning
Ashish	Data Mining
Shivam	Network Security
Shivam	Data Mining
Tarun	Network Security
Tarun	Machine Learning
Yash	Machine Learning
Yash	Data Mining

Subject –

Subject
Machine Learning
Data Mining

Suppose we want to get the students that are **NOT** learning Machine Learning or Data Mining. Then, we can perform $STUDENT(Subject) \div SUBJECT(Subject)$. This is possible since $SUBJECT(Subject) \subseteq STUDENT(Subject)$.

The result will be Shivam and Tarun.

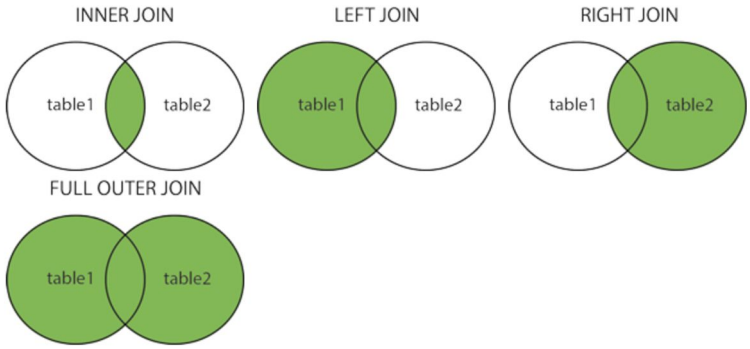
Rename Operator (ρ)

It is used to rename the relation. The format is –

$$\rho(New\ name, Old\ name)$$

TYPES OF JOINS

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



Question

Consider the following relation R(A,B,C) and S(B,D,E) with underlined primary key. The relation R contains 200 tuples and the relation S contains 100 tuples. What is the maximum number of tuples possible in the natural Join R and S?

Answer

The natural join occurs over common attributes. Hence,

$$R \bowtie S = \sigma_{R.B=S.B}(R \times S)$$

There can be a case where every value of $S.B$ has a mapping in $R.B$. Hence, the max number of tuples will be **100.** On the other hand, it can also be possible that none of the $S.B$ values are mapping with $R.B$ values. Thus, the minimum number of tuples will be **0.**

NOTE

In the Selection operator, **and** condition is represented by a ∧ and the **or** condition is represented by a ∨. For example, if we want to select tuples in relation R such that they satisfy *conditionA* ***AND*** *(condition* ***OR*** *conditionC),* then we can write –

$$\sigma_{conditionA \wedge (conditionB \vee conditionC)}(R)$$

UNION COMPATIBLE RELATIONS

2 relations are said to be Union Compatible if they satisfy the following 3 conditions –

- Both have same number of columns
- Names of attributes are same in both
- Attributes with the same name in both have the same domain

We can use the Union, Intersection and Difference operators only between Union Compatible relations.

TRANSACTIONS

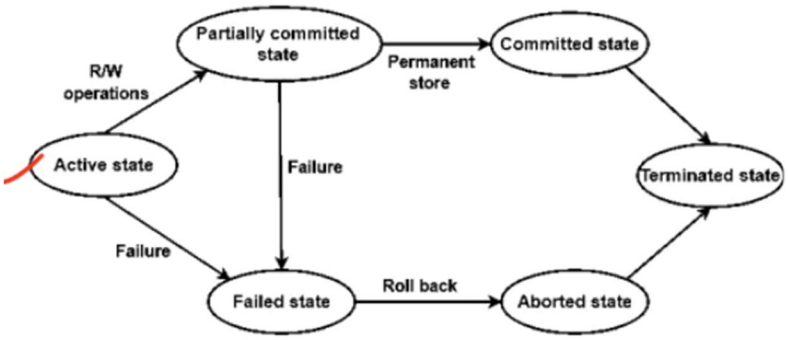
It is a set of logically related operations. There are mainly two operations in a transaction –

- **Read –** Reading from DB and storing in the buffer/main memory
- **Write –** Writes the value back to the DB from the buffer/main memory

There can be cases where the transaction starts operation but due to any error, it stops mid-way and hence causes data inconsistency. To handle such situations, we have 2 important operations –

- **Commit –** To save the changes permanently
- **Rollback –** To undo the work done

STATES OF A TRANSACTION



- **Active –** When the transaction is being executed. All changes are stored in the buffer.
- **Partially committed –** The transaction enters this state when the last instruction is executed and the changes are still in the buffer.
- **Committed –** When the changes from the buffer is stored to the DB via the commit operation, it enters the Committed state.
- **Failed –** When the transaction encounters a failure either in the active or partially committed state, it enters the failed state.
- **Aborted –** After a failure, we use the Rollback operation to undo the changes and the transaction enters the Aborted state.
- **Terminated –** This is where the transaction life cycle ends. It can be entered via committed or aborted states.

OS is responsible for monitoring and performing the transaction operations. The operations in the transaction cycle are as follows –

1) **BEGIN_TRANSACTION** → **Beginning of Transaction**
2) **READ OR WRITE** → **Read/Write operations on the DB items that are executed**
3) **END_TRANSACTION** → **End of Read/Write operations and Transaction Execution**
4) **COMMIT_TRANSACTION** → **Successful End + updates safely committed + not undone.**
5) **ROLLBACK / ABORT** → **ended unsuccessfully + updates can be undone**

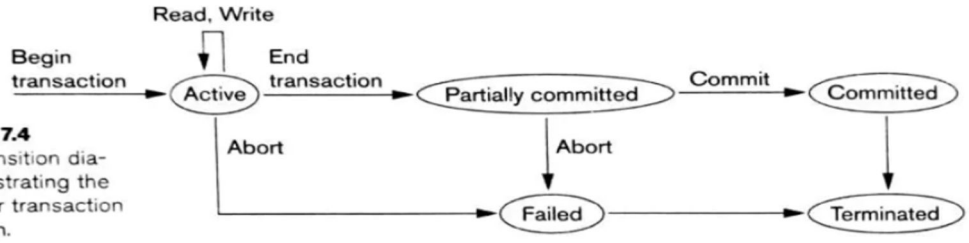


Figure 17.4
State transition diagram illustrating the states for transaction execution.

ACID PROPERTIES

These are the properties that need to be followed by the transactions to ensure DB consistency. These properties are –

- **Atomicity –** This states that the transaction must occur completely or not be executed at all. Instructions are not allowed to occur partially. Atomicity is ensured by **Transaction Control Manager.**
- **Consistency –** This property ensures that the integrity constraints of data are maintained. It is the responsibility of the **DBMS and app programmer**.
- **Isolation –** This property ensures that multiple transactions can occur simultaneously without causing any inconsistency. Transactions are executed parallelly without any interaction between the transactions. This also means that changes made by a transaction will only be visible to the other transactions when it commits the changes to the DB. It is the responsibility of **Concurrency Control manager**.
- **Durability –** This property ensures that once the transaction is completed, the data stored in the disk remains like that even if system failure occurs. This is the responsibility of **Recovery Manager.**

Question

Consider the following transaction involving two bank accounts x and y:

- **read(x);**
- **x := x – 50;**
- **write(x);**
- **read(y);**
- **y := y + 50;**
- **write(y);**

The constraint that the sum of the accounts x and y should remain constant is that of?

A. Atomicity
B. Consistency
C. Isolation
D. Durability

Answer

Option B is the correct answer

CONCURRENCY PROBLEMS

When multiple transactions execute concurrently in an uncontrolled manner, there can be several problems -

1. Dirty Read Problem
2. Unrepeatable Read Problem
3. Lost Update Problem
4. Phantom Read Problem

Dirty Read Problem

It is the situation where a transaction reads a data which has been written by an uncommitted transaction. This is because there is still a chance that the uncommitted transaction might roll back later and this will lead to data inconsistency. However, the data inconsistency only occurs when the uncommitted transaction performs a rollback.

For example,

Transaction T1	Transaction T2
R(A)	
W(A)	
	R(A) // Dirty Read
	W(A)
	Commit
Failure	

In this case, Uncommitted transaction T1 has modified the value of A but has not yet committed it. Before commit, transaction T2 reads the value of A. Later, transaction T1 fails and now performs rollback. This is a classic dirty read case for transaction 2.

Unrepeatable Read Problem

This happens when transaction gets different value when repeated reading the same variable. For example,

Transaction T1	Transaction T2
R(X)	
	R(X)
W(X)	
	R(X) // Unrepeated Read

Here, Transaction T2 has not updated the value of X but between the 2 reads, Transaction T1 has updated the value and thus T2 will read 2 different values despite it not updating shit.

Lost Update problem

This happens when multiple transactions update the same variable one after the other. For example,

Transaction T1	Transaction T2
R(A) 10	
W(A) 15	
	W(A)
	COMMIT
COMMIT	

In this case, First transaction T1 will update the value of A from 10 to 15 (say). Now, after that transaction T2 will update A to be 25. Thus, when T1 commits, it will commit 25 which was not what T1 had update. So, T1's update has been **lost.**

Thus, we can see that this problem occurs when there is a **write – write conflict.**

Phantom Read Problem

This is a variation of the Unrepeatable Read problem. This problem occurs when a transaction reads a variable once but the next time it reads, the variable has been deleted or does not exist. For example,

Transaction T1	Transaction T2
R(X)	
	R(X)
DELETE(X)	
	READ(X)

In this case, T2 first reads the value of X. However, the next time it tries to read the value of X, the variable X no longer exists as T1 has deleted it.

SCHEDULE

A transaction is a collection of operations. A collection of transactions is known as a **schedule.** It can be of 2 types –

- **Serial Schedule –** In this case, a transaction completes entire execution and only after that the next transaction can be executed. This has low throughput and resource utilization.
- **Concurrent Schedule –** There can be interleaving of transactions and a transaction can be interrupted by another transaction. This is also called a **non-serial schedule.**

T1	T2
R1(A)	
W1(A)	
	R2(A)
R1(B)	

NON- SERIAL SCHEDULES

T1	T2
R1(A)	
W1(A)	
R1(B)	
	R2(A)

SERIAL SCHEDULES

In a non-serial schedule, the transaction operations of 2 transactions can either interfere with each other or not interfere with each other. When they are not interfering, it is called a **serializable schedule.** A serializable schedule will maintain the **consistency** in the DB and behaves like a Serial schedule.

Serial Schedules	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.	Concurrency is allowed. Thus, multiple transactions can execute concurrently.
Serial schedules lead to less resource utilization and CPU throughput.	Serializable schedules improve both resource utilization and CPU throughput.
Serial Schedules are less efficient . (due to above reason)	Serializable Schedules are always better than serial schedules. (due to above reason)

Calculation of Number of Schedules : Consider there to be n number of transactions t1, t2, t3, ..., tN with N1,N2,N3,.....Nn number of operations respectively.

1. **Total Number of Schedules –**

```
(N1 + N2 + N3 + ..... + Nn)! / (N1! * N2! * N3! * ... * Nn!)
```

1. **Number of Serial Schedules –**

```
It is all possible permutations of n transactions = N!
```

RECOVERABLE vs IRRECOVERABLE SCHEDULES

Let's assume that a transaction T_j reads a value of variable X from transaction T_i and the read is a **dirty read.** Now, if T_j commits the value of X before T_i, then the schedule is called an **Irrecoverable schedule.** It is called irrecoverable because when T_j commits, there is no reverting back and we can't rollback to recover the false changes. For example –

Transaction T1	Transaction T2
R(A)	
W(A)	
	R(A) // Dirty Read
	W(A)
	Commit
Rollback	

On the other hand, if there is no dirty read of even if there is a dirty read but T_i commits before T_j, then we can perform a rollback and recover the faulty changes. Hence, these schedules are termed as **recoverable schedules.** For example –

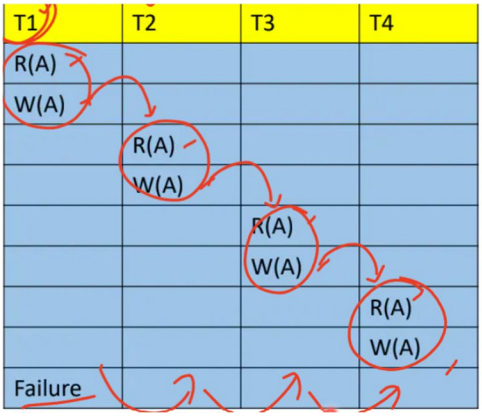
Transaction T1	Transaction T2
R(A)	
W(A)	
	R(A) // Dirty Read
	W(A)
Commit	
	Commit // Delayed

TYPES OF RECOVERABLE SCHEDULES

Recoverable schedules are classified into 3 groups –

- Cascading Schedule
- Cascadeless Schedule
- Strict Schedule

Cascading Schedule



In the example above, we can see that a failure in T1 has occurred. As a result, T2 has to rollback. Since T2 has to rollback, even T3 has to rollback. Finally, due to T3 rolling back, even T4 has to perform a rollback. Therefore, there is a **cascading rollback** that occurs and this is called a **Cascading schedule**. In case T2, T3 and T4 would have committed before T1 fails, then the schedule would have been irrecoverable.

Cascadeless Schedule

If a schedule allows only **committed read** i.e. T_j can only read after T_i commits, then the schedule is called a **Cascadeless Schedule.** Since they allow only committed read, there is no case where T_i will fail and cause a rollback of T_j. Hence, the **cascading rollback is prevented.**

T1	T2	T3
R(A)		
W(A)		
Commit		
	R(A)	
	W(A)	
	Commit	
		R(A)
		W(A)
		Commit

One important thing to note is that Cascadeless schedules allows committed reads only but can **allow uncommitted write** operations.

T1	T2
R(A)	
W(A)	
	W(A) // Uncommitted Write
Commit	

In the above example, T2 writes before T1 can commit. This is allowed by a cascadeless schedule.

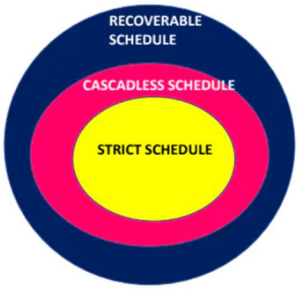
Strict Schedule

This is a sub – class of Cascadeless schedules which allow **only for committed writes**. Hence, there can be only committed reads and writes.

T1	T2
W(A)	
Commit/Rollback	
	R(A)/W(A)

NOTE

Strict schedules apply the maximum number of restrictions. Every strict schedule is a cascadeless schedule.



Question

Which Type Of Schedule ?

1. Recoverable Schedule
2. Irrecoverable Schedule
3. Both
4. None

Transaction T1	Transaction T2
R(A)	
W(A)	
	R(A) // Dirty Read
	W(A)
	Commit
Rollback	

Answer

Option 2.

SERIALIZABILITY

As mentioned previously, a non – serial schedule is said to be serializable if it maintains data consistency throughout the operations. Hence, a non – serial serializable schedule behaves like a serial schedule and has the following properties as a result –

- Data consistency is maintained
- It is recoverable
- The schedule is strict and hence also cascadeless

There are two types of serializability –

- Conflict
- View

Conflict Serializability

If a non – serial schedule can be converted into a serial schedule by swapping **non – conflicting operations**, then it is called a **conflict serializable schedule.** To understand this, we need to understand what are conflicting operations. Each conflicting operation follows the following three rules –

1. Both the operations belong to different transactions each
2. Both the operations are being performed on the same variable/data entry
3. At least one of the operations is a write operation.

For example –

T1	T2
R1(A)	
W1(A)	
	R2(A)
R1(B)	

Let us take the operations **W1(A) and R2(A).** These operations are for different transactions, being performed on the same variable and one of them is a write operation. Therefore, **they are conflicting operations.**

To figure out if a schedule is conflict serializable, we need to perform the following steps –

1. First, we draw all the transactions as nodes.
2. Then, we take a list of conflicting operations. One thing to note here is that **conflicting operations are taken in a top – down manner.** For example, in the previous example, $W1(A) \rightarrow R2(A)$ is a conflicting operation but $R2(A) \rightarrow W1(A)$ is not a conflicting operation as we don't go in a bottom – up approach.
3. Now, we can draw edges. So, for a conflicting operation $W1(A) \rightarrow R2(A)$, the edge will be drawn from T1 to T2.
4. After the edges are drawn, we have a **precedence graph.** Now we check if there are any cycles in the precedence graph. If there are **cycles, then the schedule is non – conflict serializable**. Else, it is conflict serializable.

Question

Check whether the given schedule S is conflict serializable or not-
S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

Answer

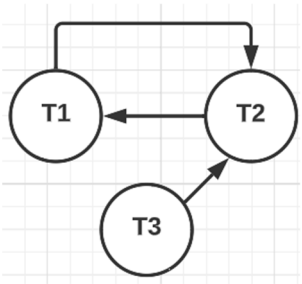
The schedule can be written as –

T1	T2	T3
R(A)		
	R(A)	
R(B)		
	R(B)	
		R(B)
W(A)		
	W(B)	

Hence, the conflicting operations are –

- $R2(A) \rightarrow W1(A)$
- $R1(B) \rightarrow W2(B)$
- $R3(B) \rightarrow W2(B)$

Thus, we can draw the precedence graph as follows –



Since there is a cycle here, the schedule is **non – conflict serializable.**

NOTE

Suppose we have two schedules S1 and S2 which have the same conflicting operations. In that case, S1 and S2 are said to be **conflict equivalent.**

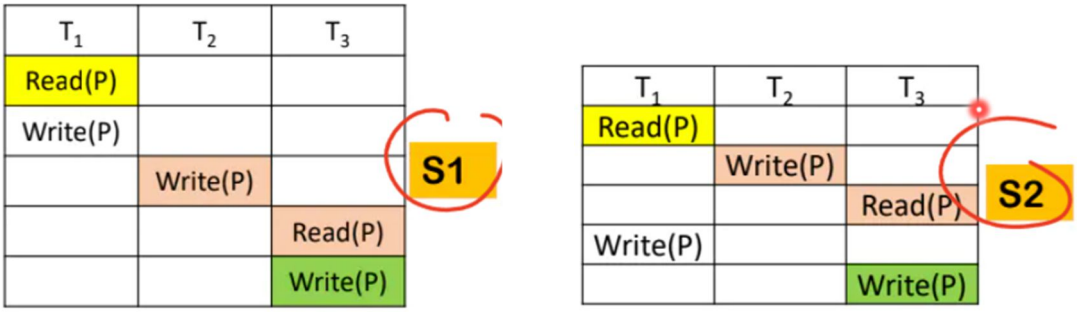
View Serializability

There can be cases where the precedence graph of a schedule results in a cycle but the schedule maintains data consistency. In short, a schedule can have a cycle in precedence graph and at the same time be serializable. Therefore, just because a schedule is not satisfying conflict serialization check doesn't mean it is non – serialization. This is where **view serialization** comes into play.

A schedule is called view serializable if it is a **view equivalent with a serial schedule**. To be view equivalent, the 2 schedules need to satisfy the following conditions –

- The same transaction must perform the first read
- The same transaction must perform the last write
- The write – read sequences must be same.

For example,



Schedule S1 is the serial schedule and S2 is the non – serial schedule. We can see that –

- Both schedules have T1 performing the initial read
- Both schedules have T3 performing the last write
- The write – read sequences are the same $(T2 \rightarrow T3)$

Hence, S1 and S2 are view equivalent and S2 is a view serializable.

Methods to check for View Serializability

1. First check if the schedule is conflict serializable. If yes, then it is also view serializable. If not, then move to Step 2.
2. Check if any transaction performs **blind write.** A blind write is when a transaction write a data value/variable without reading it first. If no, then the schedule is not view serializable. Else, move to Step 3.
3. Draw a precedence graph as follows –
    a. Draw the transactions as nodes
    b. Take the node with the initial read. Let it be T_i. Take the node which the initial write and let us label it as T_j. Draw an edge $T_i \rightarrow T_j$
    c. Take the node with the final write. Draw an edge from all the nodes to that edge.

d. Now, we can add other edges to denote write – read sequence

4. If a **cycle is present** in the graph, then the schedule is **not view serializable.** Else, it is view serializable.

Question

Consider three data items D1, D2, and D3, and the following execution schedule of transactions T1, T2 and T3. In the diagram, R(D) and W(D) denote the actions reading and writing the data item D respectively.

T1	T2	T3
	R(D3);	
	R(D2);	
	W(D2);	
		R(D2);
		R(D3);
R(D1);		
W(D1);		
		W(D2);
		W(D3);
	R(D1);	
R(D2);		
W(D2);		
	W(D1);	

(a) The schedule is serializable as T2; T3; T1;
(b) The schedule is serializable as T2; T1; T3;
(c) The schedule is serializable as T3; T2; T1;
(d) The schedule is not serializable

Answer

In this schedule, a cycle will exist between T1 and T2 thus making it **non conflict serializable.** Thus, we will now check for View serializability. We can see that none of the transactions have any blind writes. Hence, the schedule is also **not view serializable.**

Question

5.8 Consider the following schedules involving two transactions. Which one of the following statements is TRUE?

S_1 : $r_1(X); r_1(Y); r_2(X); r_2(Y); w_2(Y); w_1(X)$

S_2 : $r_1(X); r_2(X); r_2(Y); W_2(Y); r_1(Y); w_1(X)$

(a) Both S_1 and S_2 are conflict serializable
(b) S_1 is conflict serializable and S_2 is not conflict serializable
(c) S_1 is not conflict serializable and S_2 is conflict serializable
(d) Both S_1 and S_2 are not conflict serializable

[2007 : 2 Marks]

Answer

Option C

Question

5.10 Consider the following three schedules of transactions T1, T2 and T3. [Notation: In the following NYO represents the action Y (R for read, W for write) performed by transaction N on object O].

S1: 2RA 2WA 3RC 2WB 3WA 3WC
1RA 1RB 1WA 1WB
S2: 3RC 2RA 2WA 2WB 3WA 1RA
1RB 1WA 1WB 3WC
S3: 2RA 3RC 3WA 2WA 2WB 3WC
1RA 1RB 1WA 1WB

Which of the following statements is TRUE?

(a) S1, S2 and S3 are all conflict equivalent to each other
(b) No two of S1, S2 and S3 are conflict equivalent to each other
(c) S2 is conflict equivalent to S3, but not to S1
(d) S1 is conflict equivalent to S2, but not to S3

[2008 : 2 Marks]

Answer

We need to write all the conflicting operations of S1, S2 and S3. Once we do that, we will observe that the conflicting operations of S1 and S2 are the same but S3 are different. Hence, **Option D** is the correct answer.

BACKUP AND RECOVERY

The storing of a copy of the original data in case of data loss is called a **backup.** When this backed up data is used to restore the loss of data, then that process is referred to as **recovery.** The backup and recovery procedures are used to improve the DB reliability and data protection. The main reason for us to use recovery process is to combat the cases where there is a **failure before the transaction completes/commits.** Thus, we need to either Undo/Redo the changes to prevent any data inconsistency and also ensure **atomicity.**

TYPES OF FAILURES

- **System crash –** When there is a error during execution that causes entire system to crash.
- **System Error –** Programming Logical error (Divide by zero)
- **Local Error –** When conditions are such that a transaction request ends up getting cancelled.
- **Concurrency Control Enforcement –** The CCE may decide to cancel a transaction to ensure serializability and concurrency control.
- **Disk Failure –** When disks lose data due to read/write errors.
- **Catastrophe –** Extreme shit like fire, theft, natural disaster, power outage.

- **Network Failure –** Disruption in the comms between client and server.
- **Deadlock –** Two or more transactions are waiting for each other
- **Software bugs –** Bugs in the DBMS that result in unexpected outcomes
- **Power Outage –** Generator lagwa le bhadve!!!
- **Data Corruption –** Any of the above failures can cause data to become inconsistent or corrupted.

DB RECOVERY TECHNIQUES

There are mainly two techniques –

- **Undo/Rollback –** This is done in cases where the transaction hasn't committed its changes and encountered a failure. In such case, the recovery system **undoes** the changes to the last valid checkpoint.
- **Redo/Commit –** This is done for transactions that have successfully committed and then after that encounter a failure. In this case, the recovery system **redoes** or **re-applies** the changes it has made to set the DB back to the most recent stable state.

The transaction states and operations are stored in a **transaction log file** and this acts as the basis for the DB recovery system. There can also be **checkpoints** in the DB where the DB regularly stores the points where the DB was stable which are called checkpoints.

This information is needed to recover from transaction failure.

•The log is kept on disk start_transaction(T): This log entry records that transaction T starts the execution.

•read_item(T, X): This log entry records that transaction T reads the value of database item X.

•write_item(T, X, old_value, new_value): This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.

•commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.

•abort(T): This records that transaction T has been aborted.

•checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

1.Undo: using a log record sets the data item specified in log record to old value.

2.Redo: using a log record sets the data item specified in log record to new value.

The database can be modified using two approaches –

1.Deferred Modification Technique: If the transaction does not modify the database until it has partially committed, it is said to use deferred modification technique.

2.Immediate Modification Technique: If database modification occur while the transaction is still active, it is said to use immediate modification technique.

Question

Consider a simple checkpointing protocol and the following set of operations in the log.

(start, T4); (write, T4, y, 2, 3); (start, T1); (commit, T4);
(write, T1, z, 5, 7); (checkpoint); (start, T2);
(write, T2, x, 1, 9); (commit, T2); (start, T3); (write, T3, z, 7, 2);

If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list

(A) Undo: T3, T1; Redo: T2
(B) Undo: T3, T1; Redo: T2, T4
(C) Undo: none; Redo: T2, T4, T3; T1
(D) Undo: T3, T1, T4; Redo: T2

Answer

(start, T4); (write, T4, y, 2, 3); (start, T1); (commit, T4);

(write, T1, z, 5, 7); (checkpoint); (start, T2);

(write, T2, x, 1, 9); (commit, T2); (start, T3);

(write, T3, z, 7, 2);

Since T1 and T3 are not committed yet, they must be undone. The transaction T2 must be redone because it is after the latest checkpoint.

T1	T2	T3	T4
			start
			W(y,2,3)
start			
			commit
W(z,5,7)			
Checkpoint	Checkpoint	Checkpoint	Checkpoint
	start		
	W(x,1,9)		
	commit		
		start	
		W(z,7,2)	
crash	crash	crash	crash

Question

Consider the following log sequence of two transactions on a bank account, with initial balance 12000, that transfer 2000 to a mortgage payment and, then apply a 5% interest.

1. T1: start
2. T1: B old = 12000, new = 10000
3. T1: M old = 0, new = 2000
4. T1: commit
5. T2: start
6. T2: B old = 10000, new = 10500
7. T2: commit

Suppose the database system crashed just before log record 7 is written. When the system is restarted, which one statement is true of the recovery procedure?

(a) We must redo log record 6 to set B to 10500
(b) We must undo log record 6 to set B to 10000 and then redo log records 2 and 3
(c) We need not redo log records 2 and 3 because transaction T1 has committed
(d) We can apply redo and undo operations in arbitrary order because they are idempotent

[2006 : 1 Mark]

Answer

Since the system crash occurs before transaction 7, we can conclude that before the crash transaction **T1 has committed** and transaction **T2 has not committed.** Therefore, we need to **redo write ops of T1 and undo write ops of T2.** Hence, the correct answer is **Option B.**

CONCURRENCY CONTROL PROTOCOL

Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each other. The aim of concurrency control protocol is to –

- Ensure Serializability
- Prevent Deadlock
- Prevent Cascading Rollback
- Prevent Starvation
- Ensure Recoverability

There are different types of concurrency control protocols that are implemented –

- **Lock Based Protocol**
  - Basic 2-PL
  - Conservative 2-PL
  - Strict 2-PL
  - Rigorous 2-PL
- **Graph Based Protocol**
- **Time-Stamp Ordering Protocol**
- **Multiple Granularity Protocol**
- **Multi-version Protocol**

LOCK BASED PROTOCOLS

Lock is a variable that is associated with a data item that indicates the status of the data item. If we lock a data item, then no other transaction can operate on it till we unlock the data item. Simple concept. We usually deal with 2 types of locks –

- **Shared Lock (S) –** These are also referred to as **Read – Only Locks** as this lock prevents any writing or modification of the data item.
- **Exclusive Lock (X) –** These allow for both read and write operations to be performed on the data item. As a result, the Exclusive lock can be applied to a data item by a single transaction at a time and hence the name.

NOTE

A Shared Lock allows for just reading of the data item. This means that we can have **multiple transactions that can have the Shared Lock over the same data item** as multiple transactions can read the same data item without any issues. However, this is not the case with Exclusive Lock as multiple simultaneous modifications are not allowed and can cause Dirty Read, Loss Update and Phantom Read problems. Thus, we get a **compatibility matrix** as follows –

	S	X
S	Compatible	Not Compatible
X	Not Compatible	Not Compatible

Suppose we have a case where only 1 transaction is having a Shared Hold over a data item. Then, we can easily **Upgrade** the Shared lock to the Exclusive Lock $S \rightarrow X$. At the same time, there can be cases where a transaction has an Exclusive lock on the data item but we no longer have a need to write. There, we can **Downgrade** the Exclusive Lock to the Shared Lock $X \rightarrow S$.

PROBLEMS WITH SIMPLE LOCKING

Deadlock

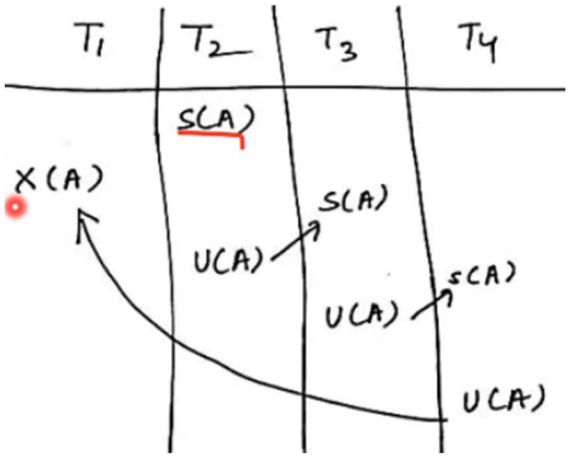
Let us take the Schedule as shown below –

	T_1	T_2
1	lock-X(B)	
2	read(B)	
3	B:=B-50	
4	write(B)	
5		lock-S(A)
6		read(A)
7		lock-S(B)
8	lock-X(A)	
9	.....	.....

In this schedule, we can see that in Operation 1, T1 applies an Exclusive Lock on B and there is no unlock. Hence, When T2 tries to apply a Shared Lock on B in Operation 7, it can't do it till T1 unlocks B. Similarly, Operation 8 of T1 can't be executed till the Shared Lock in Operation 5 applied by T2 is unlocked. Therefore, the transactions are in a state of **Deadlock.**

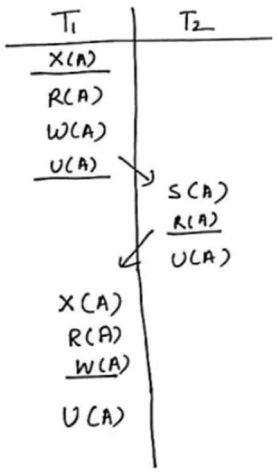
Starvation

Let us assume another schedule as follows –



In the above schedule, we can see that T2, T3 and T4 will execute first as multiple transactions can have a Shared Lock over the same data item at the same time. However, T1 will never get the Exclusive Lock over A till T4 unlocks A. Therefore, T1 will undergo **Starvation.**

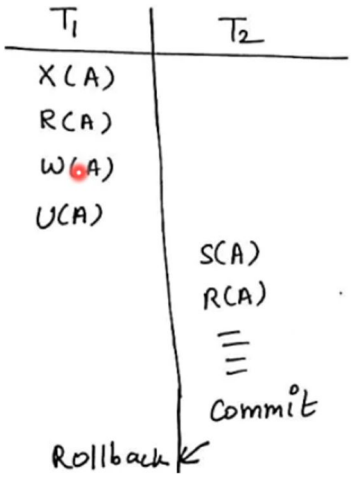
Serializability



In this case, we can see that T2 performs R(A) after T1 performs W(A) without committing. At the same time, T1 performs W(A) after T2 performs R(A) without committing. Therefore, even though we are using the simple locking system, there will be a loop in the precedence graph and the schedule is **non – serializable.**

Recoverable

Let us take the schedule as shown below –



In this case, T2 performs dirty read of A and then commits. Post that, T1 faces an error and rollbacks. Therefore, we can see that this schedule is **irrecoverable.**

TWO PHASE LOCKING

As we can see, the simple locking is not a good concurrency control protocol as it can have deadlock, starvation, irrecoverable and also does not guarantee serializability. To **ensure serializability,** we can use **2-Phase Locking (2PL).** In 2PL, we have 2 main phases (as the name suggests) –

- **Growing Phase –** In this phase, the transaction will acquire all the locks for all the data items needed. The operation where it obtains the final lock is called the **Lock Point.** This phase allows for the $S \rightarrow X$ upgradation.
- **Shrinking Phase –** This is the phase where the transaction releases all the locks it has acquired. This phase allows for the $X \rightarrow S$ down gradation.

The transaction can't release locks in the Growing phase and it can't acquire locks in the Shrinking Phase. Therefore, the schedule acts as a serial schedule as all the transactions follow the 2 phases. Therefore, **2PL ensures Serializability.**

	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 that shows how unlocking and locking work with 2-PL. Note for:
Transaction T_1:
•The growing Phase is from steps 1-3.
•The shrinking Phase is from steps 5-7.
•Lock Point at 3
Transaction T_2:
•The growing Phase is from steps 2-6.
•The shrinking Phase is from steps 8-9.
•Lock Point at 6

DRAWBACKS OF 2PL

Cascading Rollback

Take the schedule that is described below –

	T_1	T_2	T_3
1	Lock-X(A)		
2	Read(A)		
3	Write(A)		
4	Lock-S(B) --->LP	Rollback	
5	Read(B)		Rollback
6	Unlock(A),Unlock(B)		
7		Lock-X(A) ----->LP	
8		Read(A)	
9		Write(A)	
10		Unlock(A)	
11			Lock-S(A) ----->LP
12			Read(A)

FAIL Rollback

All the transactions T1, T2 and T3 follows the Growing and Shrinking phases indicating that they are following the 2PL protocol. However, we can see that once T1 fails and does a rollback, both T2 and T3 are forced to rollback as a result and therefore, we face the problem of **cascading rollback.**

Deadlock

Let us take the schedule as described below –

Schedule: Lock-X_1(A) Lock-X_2(B) Lock-X_1(B) Lock-X_2(A)

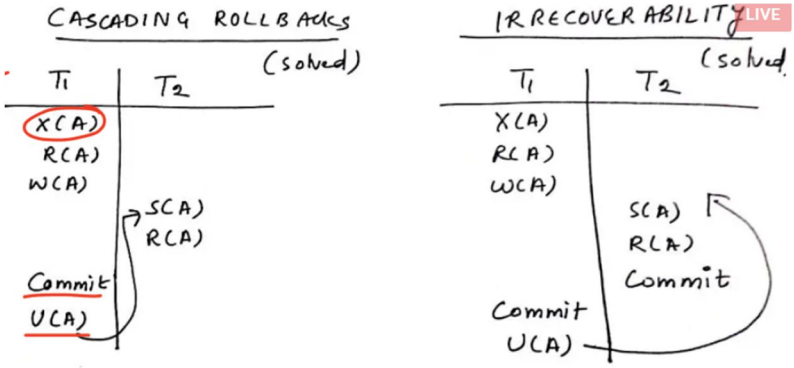
In this case, we can see that T1 can't acquire lock on B thanks to T2 and T2 can't acquire lock on A thanks to T1. Therefore, this is a classic **deadlock case.**

TYPES OF 2PL

Till now, we have discussed the **basic 2PL** protocol. We have also seen that the Basic 2PL suffers from Cascading rollback and Deadlock problems. Now, we can see the different types of enhancements done for the 2PL to ensure it would allow us to overcome the problems –

Strict 2PL

In this case, all the exclusive locks held by the transaction be released until after the transaction commits.



As we can see, the Strict 2PL will solve both the Cascading Rollback and Irrecoverable problem. However, deadlocks are still possible.

Rigorous 2PL

This is a more restrictive version of the Strict 2PL. In this case, both Exclusive and Shared locks need to be released until after the transaction commits. This also ensures that the Cascading Rollback and Irrecoverable problems are solved. However, deadlocks are still possible.

Conservative 2PL (Static 2PL)

In this case, the transaction needs to **predeclare** the **read/write sets** or operations. Once it has the read and write sets, the transaction will acquire the lock if it can get the lock for all the operations in the read or write sets. Therefore, this **solves the Deadlock** problem but it can't solve the Cascading Rollback and Irrecoverable problem.

However, Conservating 2PL is not a practical solution as it is not easy to implement a transaction where it can predeclare the read and write sets. Therefore, this is not a widely used solution.

Question

For the following schedule, implement it using the 2PL protocol and also identify the class of 2PL it belongs to.

	T_1	T_2
1	Read(A)	
2		Read(A)
3	Read(B)	
4	Write(B)	
5	Commit	
6		Read(B)
7		Write(B)
6		Commit

Answer

First thing we need to do here is to check whether the given schedule is conflict serializable or not. Since it is, we can proceed with 2PL implementation as follows –

	T_1	T_2
1	Lock-S(A)	
2	Read(A)	
3		Lock-S(A)
4		Read(A)
5	Lock-X(B)	
6	Read(B)	
7	Write(B)	
8	Commit	
9	Unlock(A)	
10	Unlock(B)	
11		Lock-X(B)
12		Read(B)
13		Write(B)
14		Commit
15		Unlock(A)
16		Unlock(B)

We can see that both Shared lock S(A) and Exclusive lock X(B) are being unlocked after the commit operation in T1 and T2. Therefore, the schedule is **Rigorous 2PL** and since all rigorous 2PL are also Strict 2PL, the above schedule is also a **Strict 2PL.**

DIFFERENCE BETWEEN CONSERVATIVE AND RIGOROUS 2PL

S.No.	Conservative 2-PL	Rigorous 2-PL
1.	A transaction has to acquire locks on all the data items it requires before the transaction begins it execution.	A transaction can acquire locks on data items whenever it requires (only in growing phase) during its execution.

2.	It does not have a growing phase.	It has a growing phase.
3.	It has a shrinking phase.	It does not have a shrinking phase.
4.	It ensures that the schedule generated would be Serializable and Deadlock-Free.	It ensures that the schedule generated would be Serializable, Recoverable and Cascadeless.
5.	It does not ensures Recoverable and Cascadeless schedule.	It does not ensures Deadlock-Free schedule.
6.	It does not ensure Strict Schedule.	It ensures that the schedule generated would be Strict.
7.	It is not used in practise as its difficult to implement.	It is easy to implement but a lighter version of it (i.e Strict 2-PL) is used in practise.
8.	In Conservative 2-PL, a transaction can read a value of uncommitted transaction.	In Rigorous 2-PL, a transaction only reads value of committed transaction.

TIME STAMP ORDERING (TSO) PROTOCOL

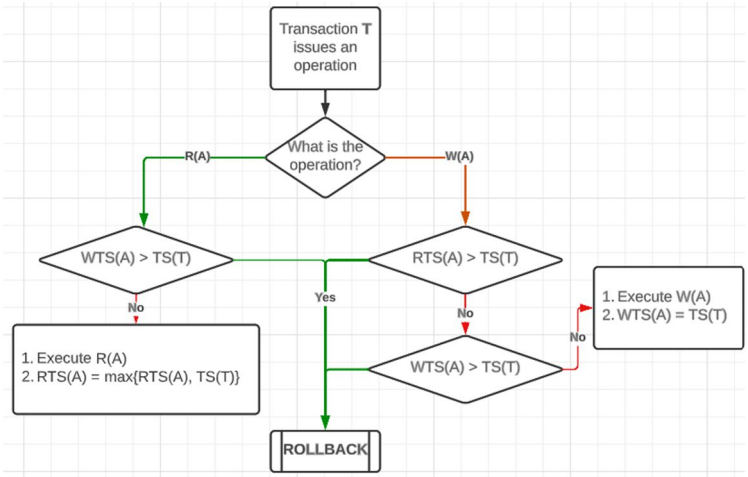
A **timestamp** is basically a tag that can be attached to any transaction which denotes the specific time on which the transaction was used. To get the timestamp, we can either take the current clock value, or we can have a logical counter that keeps incrementing and the value it returns will be the timestamp. We have 2 timestamps –

- **W-timestamp(X) –** The latest time data item X was written/updated.
- **R-timestamp(X) –** The latest time data item X was read.

As per the time stamp ordering protocol, it is ensured that the transactions must be accessed in the order of their timestamps. If we have 2 transactions $T1$ and $T2$ with the timestamps as say 100 and 200 clocks respectively, then as per TSO protocol, **$T1$ should be accessed before $T2$.** In short –

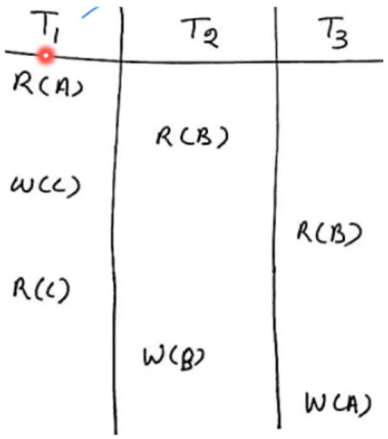
This is also referred to as the **Basic TSO protocol.** Since this protocol decides the serializability, it results in a schedule which is both **conflict and view** serializable. In addition to that, Basic TSO schedules the transaction based on timestamps which are static values aka they don't change at all. Hence, there is never a case where a new transaction comes and can execute before the pending transactions in the schedule. Therefore, Basic TSO **prevents Deadlocks.**

Whenever a transaction issue conflicting Read and Write operations, Basic TSO follows the given algorithm –



Question

Consider the schedule below –



Assume that initially the read and write timestamps for the data items are 0. Additionally, the timestamps for T1, T2 and T3 are 100, 200 and 300. Find the instruction that will be rolled back.

Answer

As we can see, there are conflicting read and write statements in this schedule. Hence, we need to use the TSO flowchart that was discussed previously. As a result, we get –

Operation 1 - $R(A)$

Transaction T1 issues Read of data item A. We can see that $WTS(A) = 0$ and $TS(T1) = 100$. Therefore, $WST(A) < TS(T1)$ and as a result, $R(A)$ operation is executed and $RTS(A) = \max\{RTS(A), TS(T1)\} = 100$. Thus,

	A	B	C
RTS	100	0	0
WTS	0	0	0

Operation 2 - $R(B)$

Transaction T2 issues Read of data item B. We can see that $WTS(B) = 0$ and $TS(T2) = 200$. Therefore, $WST(B) < TS(T2)$ and as a result, $R(B)$ operation is executed and $RTS(B) = \max\{RTS(B), TS(T2)\} = 200$. Thus,

	A	B	C
RTS	100	200	0
WTS	0	0	0

Hence, when we continue the above operations, we can see that $W(B)$ operation will have a rollback.

NOTE

TSO protocol doesn't remove the possibility of a dirty read happening and also doesn't exclude the cases where the transaction T1 will abort after T2. Hence, TSO still has **Cascading rollback and Recoverability issues.**

STRICT TSO

This is a variation of the Basic TSO where transaction T2 will delay its read/write operations until the transaction T1 who has written the value of the same data item commits or gets aborted. Hence, the schedule becomes **Strict** and also **Recoverable** and solves the **Cascading Rollback issues.**

ADVANTAGES AND DISADVANTAGES OF TSO

Advantages :

High Concurrency: Timestamp-based concurrency control allows for a high degree of concurrency by ensuring that transactions do not interfere with each other.

Efficient: The technique is efficient and scalable, as it does not require locking and can handle a large number of transactions.

No Deadlocks: Since there are no locks involved, there is no possibility of deadlocks occurring.

Improved Performance: By allowing transactions to execute concurrently, the overall performance of the database system can be improved.

Disadvantages:

Limited Granularity: The granularity of timestamp-based concurrency control is limited to the precision of the timestamp. This can lead to situations where transactions are unnecessarily blocked, even if they do not conflict with each other.

Timestamp Ordering: In order to ensure that transactions are executed in the correct order, the timestamps need to be carefully managed. If not managed properly, it can lead to inconsistencies in the database.

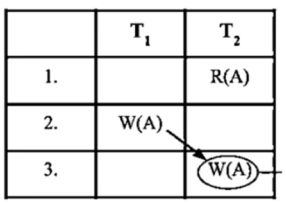
Timestamp Synchronization: Timestamp-based concurrency control requires that all transactions have synchronized clocks. If the clocks are not synchronized, it can lead to incorrect ordering of transactions.

Timestamp Allocation: Allocating unique timestamps for each transaction can be challenging, especially in distributed systems where transactions may be initiated at different locations.

THOMAS WRITE RULE

This is a variation on the Basic TSO protocol. We know that Basic TSO protocol applies only for Conflict serializable schedules. However, with Thomas Write rule, we can apply this to any view serializable schedule and it doesn't matter if the schedule is non – conflict serializable.

In the Basic TSO, we can see that when $W(A)$ is scheduled and $WTS(A) > TS(T)$, then the transaction T needs to be aborted. On the other hand, as per Thomas Write rule, the **write is ignored** and the **transaction is not aborted.** The processing continues further without any rollback. These write operations which are ignored are referred to as **Outdated/Obsolete Writes.**



Let us assume the schedule above such that $TS(T2) < TS(T1)$. As a result, we can see that $T2 \rightarrow T1$. Therefore, Operation 3 where $T1 \rightarrow T2$ is not allowed and is referred to as an Obsolete write. Basic TSO would have aborted the transaction while Thomas Write rule will ignore and move on.

In general, for a data item X, we can conclude –

- **Basic TO ProtocolNot Allowed**
  - $R_1(X)$ $W_2(X)$
  - $W_1(X)$ $R_2(X)$
  - $W_1(X)$ $W_2(X)$
- **Allowed**
  - All operations where T_2 occurs before T_1.
  - $R_1(X)$ $R_2(X)$

- **Thomas Write RuleNot Allowed**
  - $R_1(X)$ $W_2(X)$
  - $W_1(X)$ $R_2(X)$
- **Allowed**
  - All operations where T_2 occurs before T_1.
  - **Outdated Writes**: $W_1(X)$ $W_2(X)$
  - $R_1(X)$ $R_2(X)$

Question

22. State true or false: The Thomas write rule has a greater potential concurrency than the timestamp ordering protocol
a) True
b) False

Answer

In Thomas write rule, obsolete writes are ignored and the transactions are not aborted. On the other hand, Basic TSO will abort the transactions. Therefore, we can conclude that Thomas Write rule has **higher concurrency** when compared to Basic TSO. Thus, **Option A** is the correct answer.

GRAPH BASED CONCURRENCY CONTROL PROTOCOL

In this case, the transactions are represented as nodes and their conflicts are represented as the edges. The rules of the graphs are as follows –

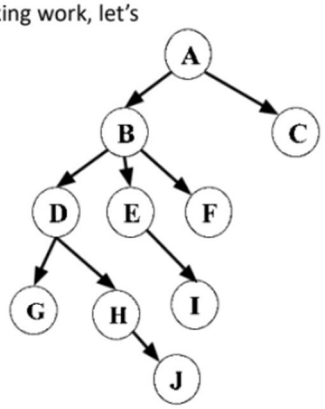
- When a transaction starts, a node is created in the graph representing the transaction.
- When a transaction accesses a data item, it acquires a shared or exclusive lock on the item, and the corresponding node is added to the graph. If a transaction tries to acquire an exclusive lock on an item that is already locked by another transaction, a conflict edge is added between the nodes representing the two transactions.
- When a transaction completes, all the edges incident on its node are removed from the graph.
- Before granting a lock to a transaction, the protocol checks the graph for any cycles. If a cycle exists, it means that there is a conflict between transactions, and one of them needs to be rolled back to break the cycle.

Graph based protocols are advantageous as they can solve complex dependencies. However, they are also computationally expensive.

- *Tree Based Protocols is a simple implementation of Graph Based Protocol.*
- A prerequisite of this protocol is that we know the order to access a Database Item. For this, we implement a **Partial Ordering** on a set of the **Database Items (D)** $\{d_1, d_2, d_3, \ldots, d_n\}$.
- The protocol following the implementation of Partial Ordering is stated as-
- If $d_i \rightarrow d_j$ then any transaction accessing both d_i and d_j must access d_i before accessing d_j.
- Implies that the set **D** may now be viewed as a directed acyclic graph (DAG), called a *database graph*.

Let's look at an example based on the above Database Graph. We have three Transactions in this schedule and this is a skeleton example, i.e, we will only see how Locking and Unlocking work, let's keep this simple and not make this complex by adding operations on data.

	T_1	T_2	T_3
1	Lock-X(A)		
2	Lock-X(B)		
3		Lock-X(D)	
4		Lock-X(H)	
5		Unlock-X(D)	
6	Lock-X(E)		
7	Lock-X(D)		
8	Unlock-X(B)		
9	Unlock-X(E)		
10			Lock-X(B)
11			Lock-X(E)
12		Unlock-X(H)	
13	Lock-X(B)		
14	Lock-X(G)		
15	Unlock-X(D)		
16			Unlock-X(E)
17			Unlock-X(B)
18	Unlock-X(G)		



From the above example, first see that the schedule is Conflict Serializable. Serializability for Locks can be written as $T_2 \rightarrow T_1 \rightarrow T_3$.
Data items Locked and Unlocked are following the same rule as given above and follow the Database Graph.

Advantage:

- Ensures Conflict Serializable Schedule.
- Ensures Deadlock Free Schedule
- Unlocking can be done anytime

With some advantages come some disadvantages also.

Disadvantage:

- Unnecessary locking overheads may happen sometimes, like if we want both D and E, then at least we have to lock B to follow the protocol.
- ***Cascading Rollbacks*** is still a problem. We don't follow a rule of when Unlock operation may occur so this problem persists for this protocol. Overall this protocol is mostly known and used for its unique way of implementing Deadlock Freedom.

CHEAT SHEET

	Conflict	View	Recoverability	Cascadless	Deadlock Freedom
TSO	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2 PL	YES	YES	YES	NO	NO
Conservative 2PL	YES	YES	NO	NO	YES
Rigorous 2PL	YES	YES	YES	YES	NO
Strict 2PL	YES	YES	YES	YES	NO

Question

Consider the following two statements about database transaction schedules:

I. Strict two-phase locking protocol generates conflict serializable schedules that are also recoverable.

II. Timestamp-ordering concurrency control protocol with Thomas Write Rule can generate view serializable schedules that are not conflict serializable.

Which of the above statements is/are TRUE?

A Both I and II

B I only

C II only

D Neither I nor II

GATE 2019

Answer

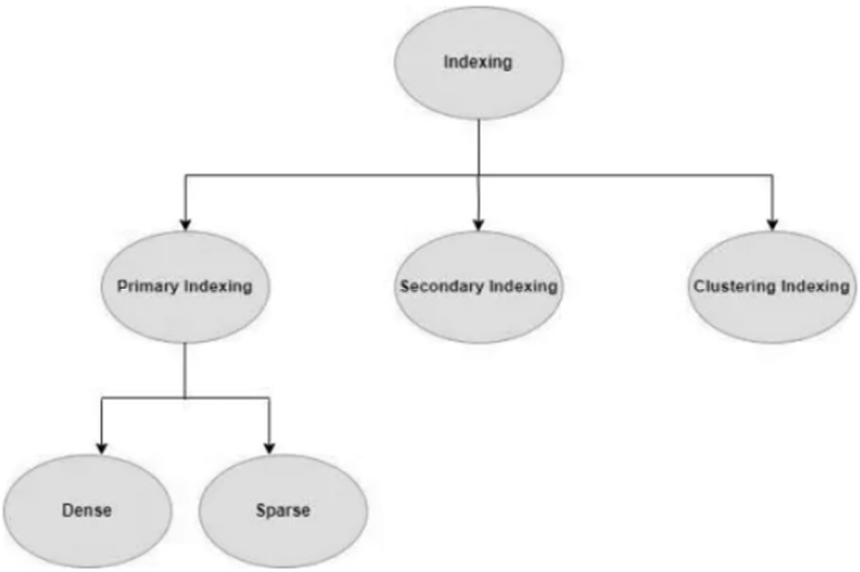
Option A

INDEXING IN DB

Indexing is a way to minimize the disk accesses required by the DB. This helps to quickly access the data and prevent unnecessary overhead. An index basically has 2 parts –

- **Search Key –** The first part is the search key which is either a primary/candidate key for which we need to get the data.
- **Data Reference/Pointer –** This is the set of pointers holding the address of the disk block where the data can be found

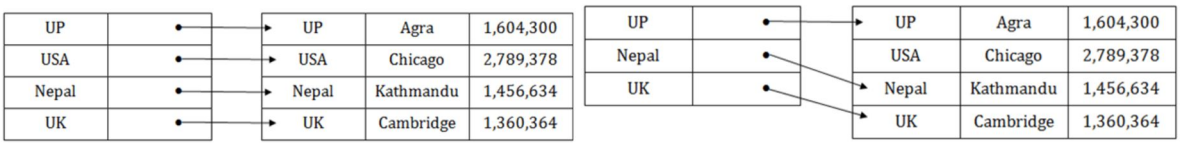
TYPES OF INDEXING



Primary Indexing

This is when the primary key is used as the index. Since the PK is always unique and ordered, the indexing will be **in ascending order** and at the same time will be **1:1 mapping.** Since the indexes are ordered, it is efficient to search. There are two types of Primary Indexing –

- **Dense Index –** In this indexing method, each value in the data file, there exists an index.
- **Sparse Index –** In this case, only select values in the data file have a corresponding index mapping. To get the disk address for the values that don't have an index map, we go to the nearest value with index map and then parse the disk sequentially after that till we get the desired value.



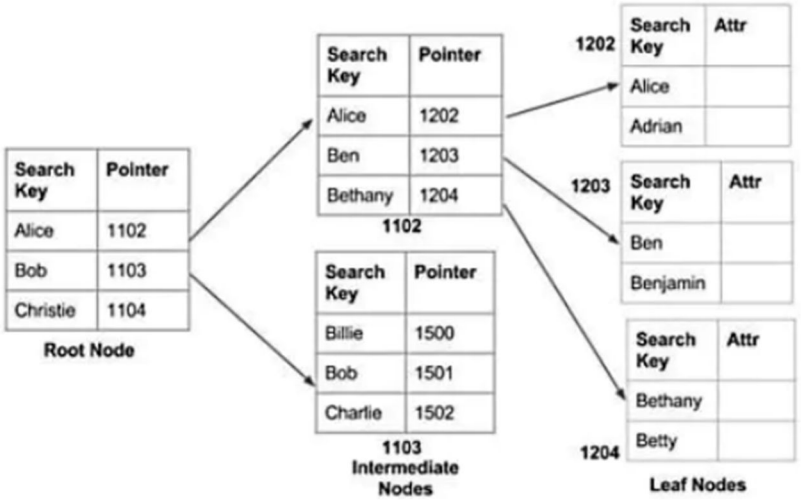
Dense Index *Sparse Index*

Clustered Indexing

When two or more records are stored in the same file location, then it is called **Clustered Indexing.** There are cases where we need to perform clustering based on non – primary key attributes. In such cases, we will have to combine multiple attributes and then perform the cluster Indexing. The output of clustered indexing is **always in ascending order.**

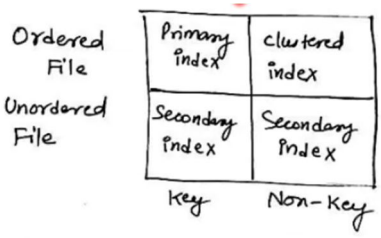
Non – Clustered/Secondary Indexing

In this case, the data is not present in the clusters. Instead, the virtual references of the data are present in the clusters. Here, the data is not sorted but the references are ordered. Since the data is not physically stored in the clustering, we need to perform more operations to get the data in this case when compared to Clustered indexing. For example –



CLUSTERED INDEX	NON-CLUSTERED INDEX
Clustered index is faster.	Non-clustered index is slower.
Clustered index requires less memory for operations.	Non-Clustered index requires more memory for operations.
In clustered index, index is the main data.	In Non-Clustered index, index is the copy of data.
A table can have only one clustered index.	A table can have multiple non-clustered index.
Clustered index has inherent ability of storing data on the disk.	Non-Clustered index does not have inherent ability of storing data on the disk.
Clustered index store pointers to block not data.	Non-Clustered index store both value and a pointer to actual row that holds data.
In Clustered index leaf nodes are actual data itself.	In Non-Clustered index leaf nodes are not the actual data itself rather they only contains included columns.
In Clustered index, Clustered key defines order of data within table.	In Non-Clustered index, index key defines order of data within index.
A Clustered index is a type of index in which table records are physically reordered to match the index.	A Non-Clustered index is a special type of index in which logical order of index does not match physical stored order of the rows on disk.

In general, we can classify the indexing methods as follows –

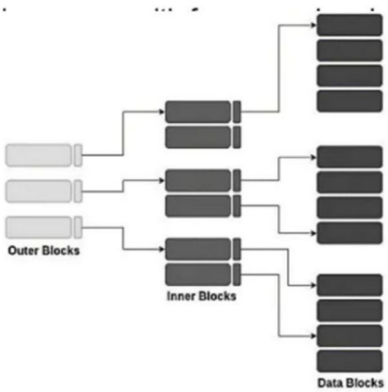


INDEXING ATTRIBUTES

- **Access Types**: This refers to the type of access such as value based search, range access, etc.
- **Access Time**: It refers to the time needed to find particular data element or set of elements.
- **Insertion Time**: It refers to the time taken to find the appropriate space and insert a new data.
- **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead**: It refers to the additional space required by the index.

MULTILEVEL INDEXING

When the size of the database grows, the indices also grow. Thus, a single index will be too large to be stored in the main memory. To solve this, we use **multilevel indexing** where we have outer blocks and inner blocks.



FILE ORGANIZATION

A file is a collection of related information on the secondary memory. A File Organization refers to the logical relationships between the files to help us access the files. There are five types of file organizations –

- Sequential
- Heap
- Hash
- B+ Tree
- Clustered

Sequential File Organization

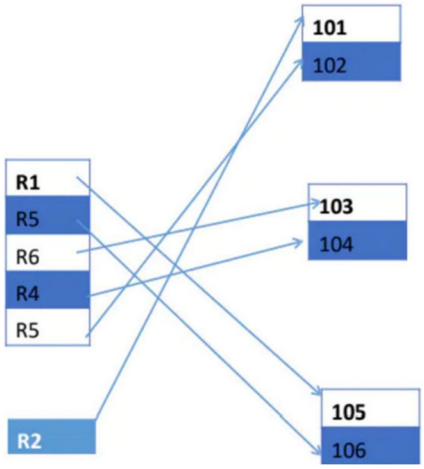
This is the easiest method where the files are stored in a sequence. There are 2 ways to implement this method –

- **Pile File Method –** In this case, the files are stored in the same sequence in which they are added.
- **Sorted File Method –** In this case, the files are added and then sorted in the ascending order.

The Sequential File Organization is simple and cheap, but at the same time is takes a long time to access a file (sequential searching) and will have the highest space for storage.

Heap File Organization

In this case, the files are stored in blocks. There is no sorting and the DBMS has the responsibility to take care of the mapping.



This is good for smaller DBs but at the same time there are a lot of unused memory blocks.

Hash File Organization

In this case, the index can be found by using a **hash function.** The records are stored in memory locations which are called **Data buckets.** There can be 2 types of hashing in DBMS –

- **Static –** In this case, the number of data buckets remains the same and the hash function also remains the same. So, the indexing result is always the same.
- **Dynamic –** In Static hashing, the number of data buckets remain the same. To accommodate the cases where the data buckets increase, we resort to dynamic hashing.

BUCKET OVERFLOW

In this case, the hash function produces the block which is already occupied. For this, there can be 3 solutions –

- **Open hashing –** Where the next free data block is filled
- **Chaining/Closed Hashing –** As studied in P&DS, we create linked lists to map to the already occupied block
- **Dynamic Hashing –** Just use dynamic hashing and increase the number of data buckets.

BALANCED TREE (B – TREE)

These are **self – balancing trees** which are used to store and manage large data sets and helps to simplify the scenarios where **Multilevel Indexing** has been implemented. It is called a **balanced tree** because all the **leaf nodes** are on the **same level.**

In a B – tree, every node can have **multiple keys** in it. Each node is made up of the following three elements –

- **Keys –** These are the keys that help to perform the indexing.
- **Record Pointer (RP) –** Each key has a corresponding record pointer which points to the block in the secondary memory where the required data for that key is stored.
- **Block/Tree Pointer (BP) –** It is the pointer that points to the child nodes for that node in the B – tree.

Hence, we can see that for any node in the B – tree –

$$No\ of\ keys = No\ of\ RP$$

$$No\ of\ children = No\ of\ BP$$

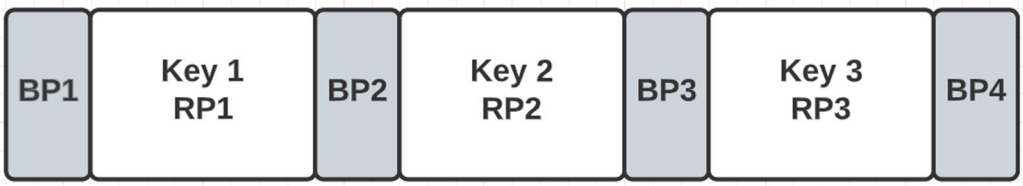
The **max no of children** any node can have is termed as the **order** of the B-tree. Therefore, when we say that if a B-tree has an order of 4, it means that any node can have a maximum of 4 children. Additionally, for any node we have –

$$No\ of\ children = No\ of\ keys + 1$$

Given this information, let us assume a B-tree of order $\boldsymbol{P}$**.** Then –

	ROOT NODE	OTHER NON-LEAF NODE
Min no of children	2	$ceil(P/2)$
Max no of children	P	P
Min no of keys	1	$ceil(P/2) - 1$
Max no of keys	$P - 1$	$P - 1$

Thus, for an order 4 B-tree, the node with max number of children will look like –



From here also, we can see that the number of keys is 1 less than the number of children.

NOTE

Suppose we have $\boldsymbol{n}$ nodes and the order of the B-tree is $\boldsymbol{m}$, then we can write –

$$Min\ tree\ height = h_{min} = ceil(\log_m(n+1)) - 1$$

$$Max\ tree\ height = h_{max} = floor\left[\log_{ceil(\frac{m}{2})}\left(\frac{n+1}{2}\right)\right]$$

B-TREE TRAVERSAL

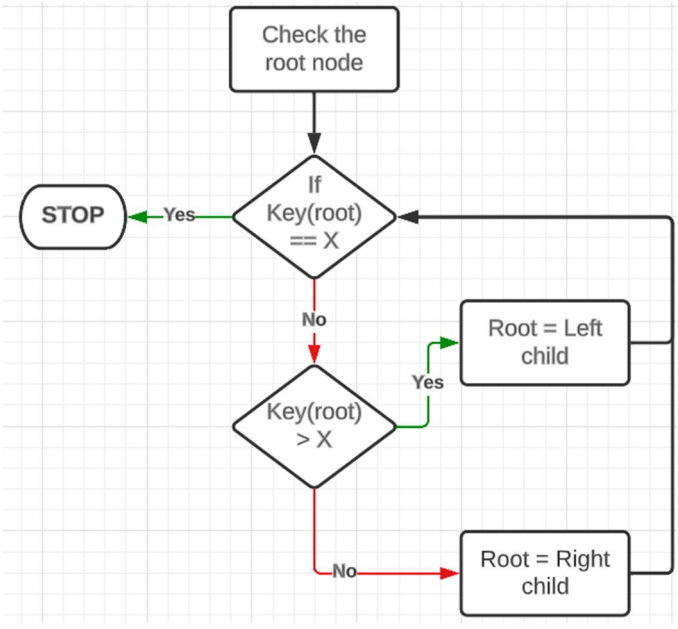
A B – tree follows the **Inorder traversal** where we first print the left child, then the root and finally the right child.

B-TREE SEARCHING

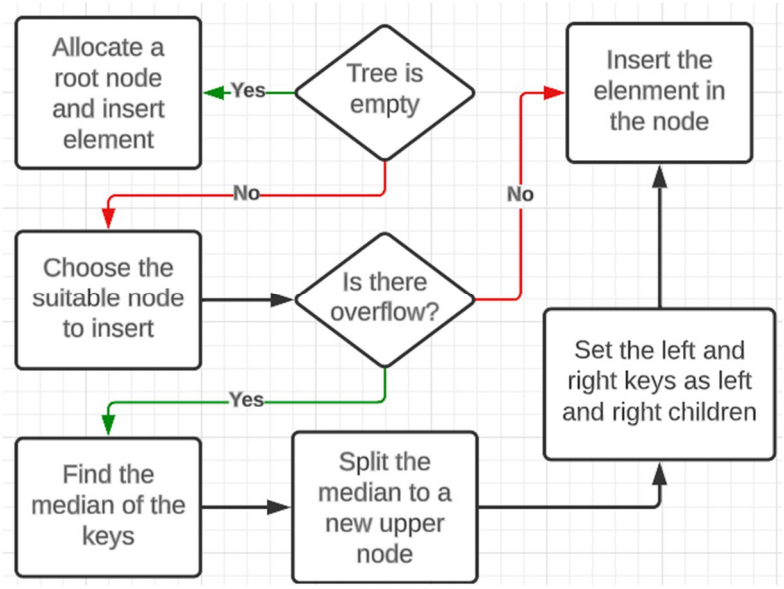
We know that **keys in a node are ordered** and at the same time, we have –

$$Keys\ in\ left\ child < Keys\ in\ root < Keys\ in\ right\ child$$

Therefore, for searching an element $\boldsymbol{X}$ we can use the following algorithm –



B – TREE INSERTION



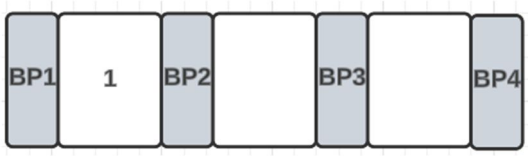
In the above case, **overflow** refers to the case where the node is trying to fit more than the max number of keys it can. Let us take an example as follows –

$$Keys = \{1,2,3,4,5,6,7,8,9,10\}$$

$$Order = 4$$

For this case, we can have maximum of **3 keys per node.** Let us begin now 😊

Initially, there are no nodes and keys inserted. Thus, we create a root node and allocate the element 1 to it as follows –



Once this is done, we can also insert keys 2 and 3 in the same node as there will be no overflow –

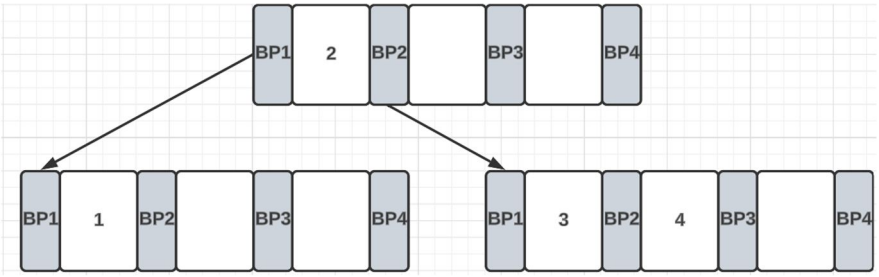
Now, we can try to insert key 4 to the node. In this case, there is an overflow. Thus, we need to find the **median** of the keys and push the median as the root node and split the other keys as the left and right children.

$$Median(1,2,3,4) = 2 \; or \; 3$$

We shall take 2. Thus, we keep key 2 as the root node and have –

$$Left \; children = \{1\}$$

$$Right\ children = \{3,4\}$$



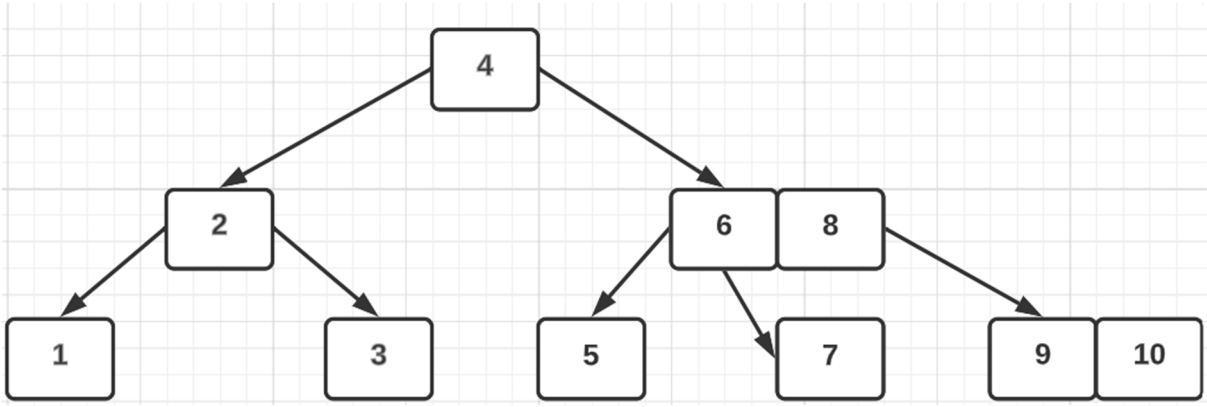
Now, we can add key 5 to the right child of the root node without any overflow. However, when it comes to adding key 6, we will face overflow and must repeat the process again –

$$Median(3,4,5,6) = 4$$

$$Left\ children = \{3\}$$

$$Right\ children = \{5,6\}$$

Hence, Key 4 is now pushed to the root level. We can continue this entire process and end up with the following B – tree –



B – TREE DELETION

1. **If the node to delete is a leaf node:**

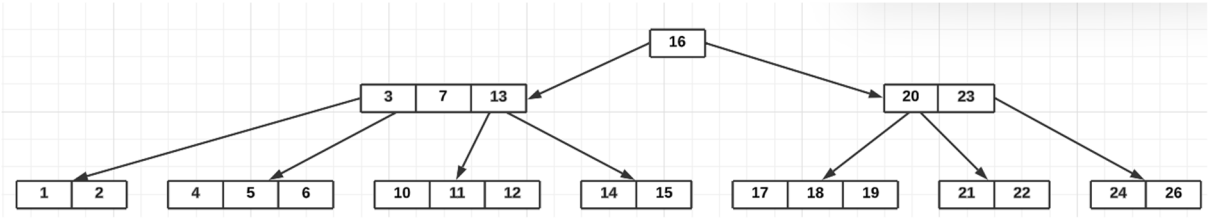
   a. Locate the leaf node containing the desired key.
   b. If the node has at least $\lfloor \frac{m}{2} \rfloor$ keys, delete the key from the leaf node.
   c. If the node has less than $\lfloor \frac{m}{2} \rfloor$ keys, take a key from its right or left immediate sibling nodes. To do so, do the following:
      i. If the left sibling has at least $\lfloor \frac{m}{2} \rfloor$ keys, push up the in-order predecessor, the largest key on the left child, to its parent, and move a proper key from the parent node down to the node containing the key; then, we can delete the key from the node.
      ii. If the right sibling has at least $\lfloor \frac{m}{2} \rfloor$ keys, push up the in-order successor, the smallest key on the right child, to its parent and move a proper key from the parent node down to the node containing the key; then, we can delete the key from the node.

iii. If the immediate siblings don't contain at least $\lfloor \frac{m}{2} \rfloor$ keys, create a new leaf node by joining two leaf nodes and the parent node's key.
iv. If the parent is left with less than $\lfloor \frac{m}{2} \rfloor$ keys, apply the above process to the parent until the tree becomes a valid B-Tree.

2. **If the node to delete is an internal node:**

a. Replace it with its in-order successor or predecessor. Since the successor or predecessor will always be on the leaf node, the process will be similar as the node is being deleted from the leaf node.

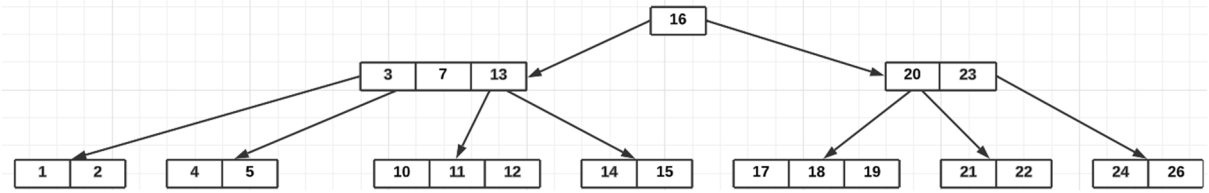
For example,



In the above B-tree, the minimum number of keys in a node can be **2.** Let us perform the deletion operations –

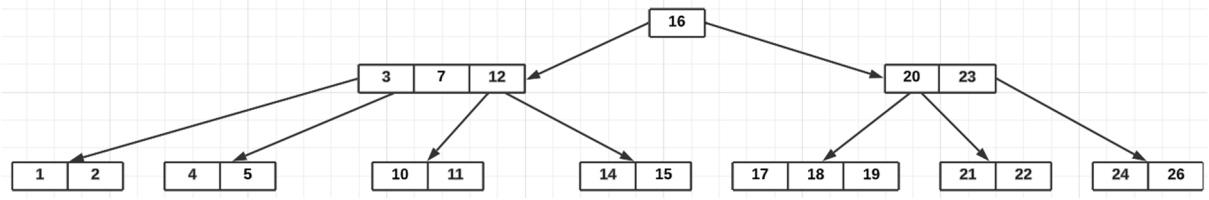
Delete Key 6

We can see that Key 6 is in a leaf node and at the same time the node doesn't go into **underflow.** Therefore, we can simply delete 6.



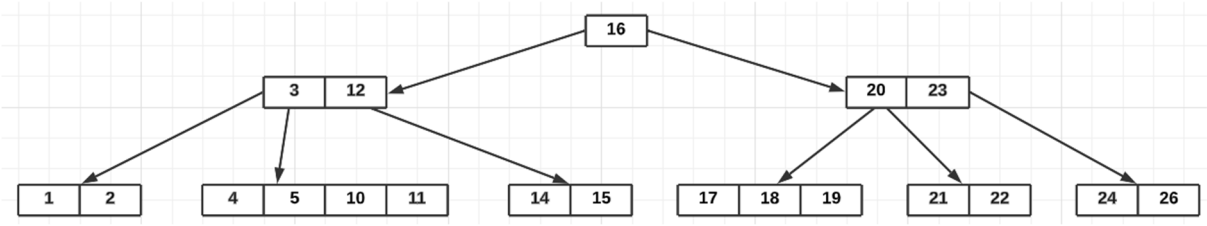
Delete Key 13

We can see that key 13 is an intermediate node and thus, we can't simply delete it. To delete it, we can go for **pushing the largest predecessor (12)** to the root level. We can't push the smallest successor (14) to the root level as the number of keys need to be at least 2 in each node. Therefore, we get –



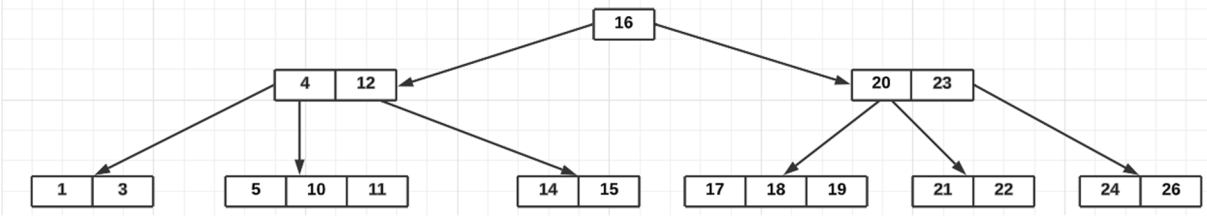
Delete Key 7

Again, key 7 is an intermediate node. In this case, we can't take the predecessor or successor as at least 1 node will be left with a single key in either scenario. Hence, we will **merge** the two leaf nodes and get the following tree –



Delete Key 2

Here key 2 is a leaf node but we can't simply delete it as it will cause the node to have just 1 key left. Hence, we take the successor from the root node (3) and to replace 3, we take the successor from the right child (4) into the root node.



NOTE

In all cases, we can say for a B – tree –

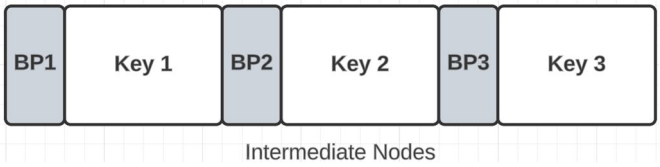
$$Time\ complexity = O(\log n)$$

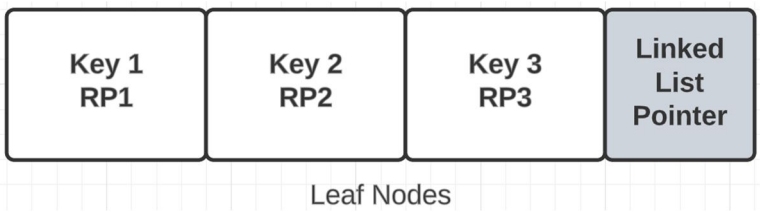
$$Space\ complexity = O(n)$$

B+ TREE

These are an enhanced version of the B – trees wherein the **intermediate nodes don't have the record/data pointers.** Only the **leaf nodes have the record/data pointer.** In short, the intermediate nodes only have keys that help in navigating and indexing through the B+ tree. Only the leaf nodes will store the RPs and can be used to access the disk storage.

One more important thing is that the leaf nodes **are arranged as a linked list** to ensure there is sequential access of the RPs and disk data. Therefore, for a B+ tree, we can see –





One more thing to note here is that since we are not storing the RPs at every node, we can save up on space and also have a larger number of leaf nodes which ensures better indexing. In addition to that, there can be cases where a key is repeated in an intermediate node and also in a leaf node.

Other than this, B+ and B trees are both **balanced, ordered and multi-level.**

Parameters	B+ Tree	B Tree
Structure	Separate leaf nodes for data storage and internal nodes for indexing	Nodes store both keys and data values
Leaf Nodes	Leaf nodes form a linked list for efficient range-based queries	Leaf nodes do not form a linked list
Order	Higher order (more keys)	Lower order (fewer keys)
Key Duplication	Typically allows key duplication in leaf nodes	Usually does not allow key duplication
Disk Access	Better disk access due to sequential reads in linked list structure	More disk I/O due to non-sequential reads in internal nodes
Applications	Database systems, file systems, where range queries are common	In-memory data structures, databases, general-purpose use
Performance	Better performance for range queries and bulk data retrieval	Balanced performance for search, insert, and delete operations
Memory Usage	Requires more memory for internal nodes	Requires less memory as keys and values are stored in the same node

Question

Which one of the following is used to represent the supporting many-one relationships of a weak entity set in an entity-relationship diagram ?
(A) Diamonds with double/bold border
(B) Rectangles with double/bold border
(C) Ovals with double/bold border
(D) Ovals that contain underlined identifiers

GATE 2020

Answer

Option A is correct

Question

A many-to-one relationship exists between entity sets r 1 and r 2 . How will it be represented using functional depedencies if Pk(r) denotes the primary key attribute of relation r ?
(A) $Pk(r_1) \rightarrow Pk(r_2)$
(B) $Pk(r_2) \rightarrow Pk(r_1)$
(C) $Pk(r_2) \rightarrow Pk(r_1)$ and $Pk(r_1) \rightarrow Pk(r_2)$
(D) $Pk(r_2) \rightarrow Pk(r_1)$ or $Pk(r_1) \rightarrow Pk(r_2)$

UGC NET 2018

Answer

We know that $r1 \rightarrow r2$ is a many to one relationship. So, multiple records in $r1$ will be mapped to the same entity in $r2$. However, if we have an entity in $r2$, there are multiple entities in $r1$ that can be mapped to it. Therefore, the $r1$ entity will be able to uniquely identify an entity in $r2$. Hence, **Option A** is the correct answer

Question

What kind of mechanism is to be taken into account for converting a weak entity set into strong entity set in entity-relationship diagram
1.Generalization
2.Aggregation
3.Specialization
4.Adding suitable attributes

UGC NET 2014

Answer

To convert a weak entity set to a strong entity set, we need to combine the primary key of the strong set with the discriminator from the weak set. Hence, **Option 4** is the correct answer

Question

In E-R model, Y is the dominant entity and X is subordinate entity
(A) If X is deleted, then Y is also deleted
(B) If Y is deleted, then X is also deleted
(C) If Y is deleted, then X is not deleted
(D) None of the above

Answer

Since Y is the dominant entity and it has a subordinate as X, deleting Y will also delete X. Hence, **Option B** is correct.

Question

In an Entity-Relationship (ER) model, suppose R is a many-to-one relationship from entity set E1 to entity set E2. Assume that E1 and E2 participate totally in R and that the cardinality of E1 is greater that the cardinality of E2. Which one of the following is true about R?

GATE 2018

(A) Every entity in E1 is associated with exactly one entity in E2.

(B) Some entity in E1 is associated with more than one entity in E2.

(C) Every entity in E2 is associated with exactly one entity in E1.

(D) Every entity in E2 is associated with at most one entity in E1.

Answer

As per the question –

- $E1 \rightarrow E2$ is a many to one relationship. This means that multiple elements in $E1$ are mapped to 1 element in $E2$.
- We can also see that $E1$ participates totally. That means, each element in $E1$ has a mapping.

Therefore, from the above statements we can conclude that the relationship allows for every element in $E1$ to be mapped to a single element in $E2$. Thus, **Option A** is correct.

Question

For a weak entity set to be meaningful, it must be associated with another entity set in combination with some of their attribute values, is called as:

UGC NET 2015

1.Neighbour Set

2.Strong Entity Set

3.Owner Entity Set

4.Weak Set

Answer

The answer is **Option 3** since Owner Entity set is another name for **Identifying Set.**

Question

Which of the following statements is FALSE about weak entity set?

UGC NET 2015

(A) Weak entities can be deleted automatically when their strong entity is deleted.

(B) Weak entity set avoids the data duplication and consequent possible inconsistencies caused by duplicating the key of the strong entity.

(C) A weak entity set has no primary keys unless attributes of the strong entity set on which it depends are included

(D) Tuples in a weak entity set are not partitioned according to their relationship with tuples in a strong entity set.

Answer

Since Weak Entity set is dependent on Strong Entity set, the deletion of the Strong entity will cause the weak entity to be deleted. Hence, **Option A** is **TRUE.** Also, the weak entity set forms its keys by taking the Primary Key of the strong entity set and the discriminator. Since Primary keys are all unique, there will no duplication, redundancy or inconsistency in the data of the weak entity as well. Hence, **Option B is TRUE.** Since we know that weak entity set doesn't have a primary key, **Option C is also TRUE.**

Thus, the incorrect statement here is **Option D.**

Question

Immunity of the external schemas (or application programs) to changes in the conceptual schema is referred to as:
(A) Physical Data Independence
(B) Logical Data Independence
(C) Both (a) and (b)
(D) None of the above

ISRO 2018

Answer

The immunity to change is called Data Independence. The data independence between External schema (View) and Conceptual Schema is referred to as **Logical Data Independence** making the correct option to be **Option B**

Question

Under normal circumstances , the cardinality ratio of binary relationship "Write" relating "author" and "book" entities is :
1. 1:1
2. N:n
3. N:1
4. 1:n

CIL PART B

Answer

We know that one author can write multiple books and at the same time, one book may require the contribution of multiple authors. Therefore, this is a **many-to-many relationship** which in turn makes **Option 2** as correct.

Question

Which of the following is true ?

CIL PART

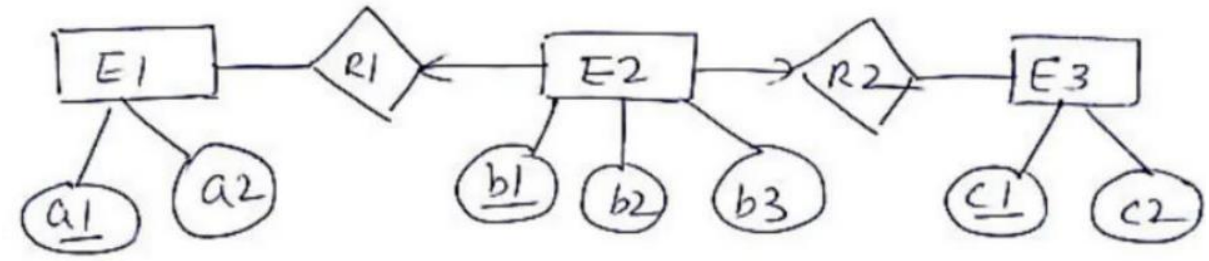
1. Cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate it.
2. Cardinality ratio for a binary relationship specifies the minimum number of relationship instances that an entity can participate it.
3. The partial participation constraint is called existence dependency
4. Cardinality ratio for a binary relationship specifies the average number of relationship instances that an entity can participate it.

Answer

Cardinality gives the information of the **max** number of times an entity can participate in the relationship. Hence, **Option 1** is correct.

Question

Find the number of tables needed to represent this ER model



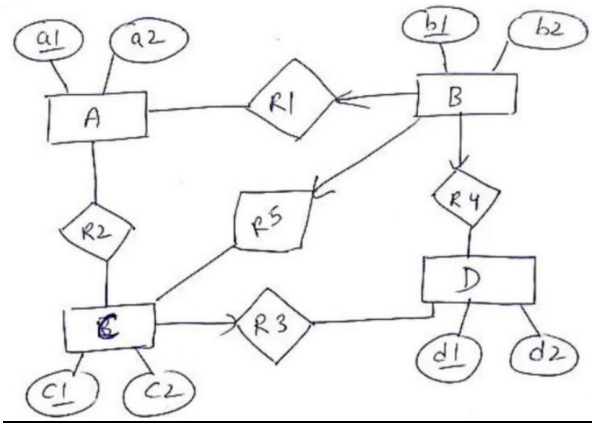
Answer

$$E1 = (\underline{a1}, a2)$$

$$E2R1R2(\underline{a1}, \underline{b1}, b2, b3, \underline{c1})$$

$$E3(\underline{c1}, c2)$$

Question



Answer

$$A(\underline{a1}, a2)$$

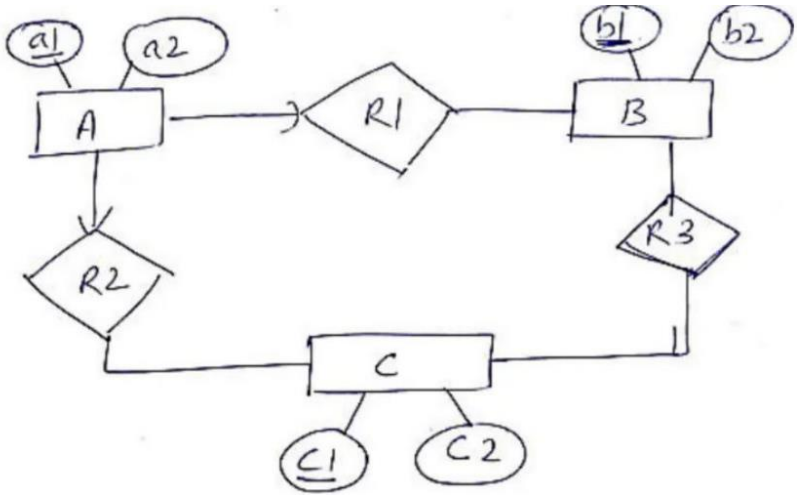
$$BR1R5R4(\underline{b1}, b2, \underline{a1}, \underline{c1}, \underline{d1})$$

$$R2(\underline{a1}, \underline{c1})$$

$$CR3(\underline{c1}, c2, \underline{d1})$$

$$D(\underline{d1}, d2)$$

Question



Answer

$$AR1R2(\underline{a1}, a2, \underline{c1}, \underline{b1})$$

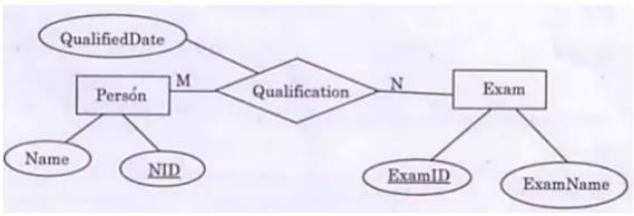
$$B(\underline{b1}, b2)$$

$$C(\underline{c1}, c2)$$

$$R3(\underline{b1}, \underline{c1})$$

Question

Which of the following possible relations will not hold if the above ERD is mapped into a relation model?
(A) Person (NID, Name)
(B) Qualification (NID, ExamID, QualifiedDate)
(C) Exam (ExamID, NID, ExamName)
(D) Exam (ExamID, ExamName)



Answer

The tables here will be –

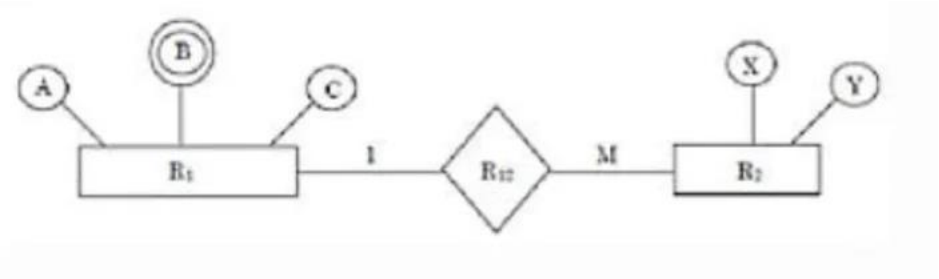
$$Person(\underline{NID}, Name)$$

$$Exam(\underline{ExamID}, ExamName)$$

$$Qualification(\underline{NID}, \underline{ExamID}, QualifiedDate)$$

Hence, **Option C** is the correct option.

Question



Minimum number of tables required to convert ER diagram to tables ?

1. **2**
2. **4**
3. **3**
4. **5**

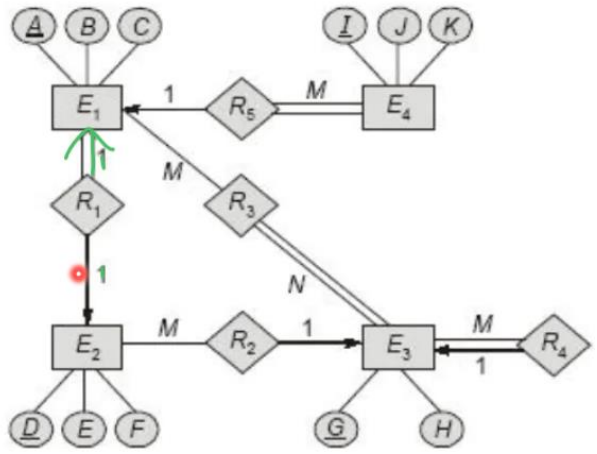
Answer

$$R_2R_{12}(X, Y, A)$$

$$R_1(A, B)$$

$$R_2(A, C)$$

Question

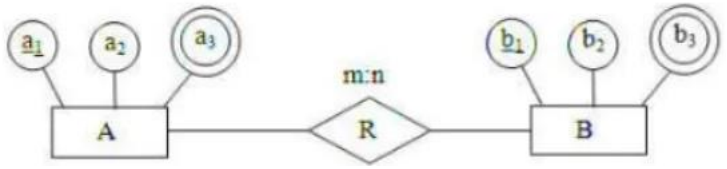


Answer

The following tables will be needed –

- E1R1E2R2 (one-to-one with one partial and one total participation)
- E3R4
- R3
- E4R5

Question



Answer

$$A(\underline{a1}, a2)$$

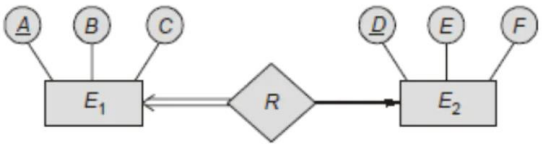
$$A(\underline{a1}, a3)$$

$$R(\underline{a1}, \underline{b1})$$

$$B(\underline{b1}, b2)$$

$$B(\underline{b1}, b3)$$

Question



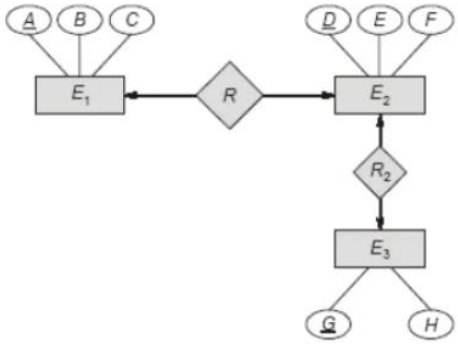
If 'n' entries in E1 and 'm' entries in E2. How many entries in relationship set (R)?
1.At least n
2.At most n
3.Exactly n
4.At least n and atmost m

Answer

Since there are 2 arrows, we can see that this is a one-to-one relationship. At the same time, we can see that E1 has total participation. That means, all n entries in E1 are being mapped to a single element in E2. Therefore, there will be **exactly n** entries in the relationship set and hence **Option 3** is the correct answer.

Question

Find the minimum number of tables required to represent this ER model.



Answer

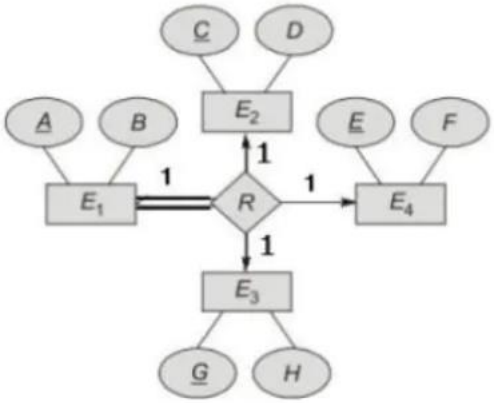
$$E1R(\underline{A}, B, C, \underline{D})$$

$$E2R2(\underline{D}, E, F, \underline{G})$$

$$E3(\underline{G}, H)$$

Question

Find the minimum number of tables required to represent this ER model.



Answer

$$E_2(\underline{C}, D)$$

$$E_1RE_4(\underline{A}, B, \underline{E}, F)$$

$$E_3(\underline{G}, H)$$

Question

Consider the following entity relationship diagram (ERD), where two entities E1 and E2 have a relation R of cardinality 1 : m

The attributes of E1 are A11, A12 and A13 where A11 is the key attribute. The attributes of E2 are A21, A22 and A23 where A21 is the key attribute and A23 is a multi-valued attribute. Relation R does not have any attribute. A relational database containing minimum number of tables with each table satisfying the requirements of the third normal form (3NF) is designed from the above ERD. The number of tables in the database is

(a) 2 (b) 3
(c) 5 (d) 4

Answer

$$E1(\underline{A11}, A12, A13)$$

$$E21R(\underline{A21}, A22, A11)$$

$$E22R(\underline{A21}, A23, A11)$$

Question

Consider the entities 'hotel room', and 'person' with a many to many relationship 'lodging' as shown below:

If we wish to store information about the rent payment to be made by person (s) occupying different hotel rooms, then this information should appear as an attribute of

(a) Person (b) Hotel Room
(c) Lodging (d) None of these

Answer

We need to store the information about the rent being paid by a person lodging in a particular hotel room. Since, we need both the hotel room and person details, the attribute rent should belong to **lodging** as it maps the person to the hotel room. Hence, **Option C** is the correct answer.

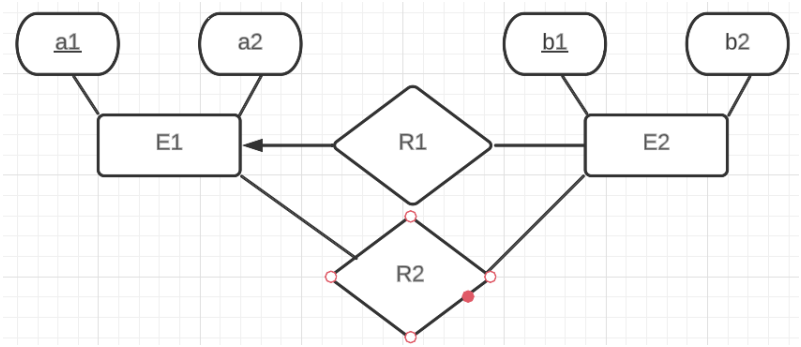
Question

Let E_1 and E_2 be two entities in an E/R diagram with simple single-valued attributes. R_1 and R_2 are two relationships between E_1 and E_2, where R_1 is one-to-many and R_2 is many-to-many. R_1 and R_2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model?

(a) 2 (b) 3
(c) 4 (d) 5

Answer

For this case, we have –



$$E1(\underline{a1}, a2)$$

$$R2(\underline{a1}, \underline{b1})$$

$$E2R1(\underline{b1}, b2, \underline{a1})$$

Hence, the correct answer is **Option B.**

Question

Consider an Entity-Relationship (ER) model in which entity sets E_1 and E_2 are connected by an m : n relationship R_{12}. E_1 and E_3 are connected by a 1 : n (1 on the side of E_1 and n on the side of E_3) relationship R_{13}.

E_1 has two single-valued attributes a_{11} and a_{12} of which a_{11} is the key attribute. E_2 has two single-valued attributes a_{21} and a_{22} of which a_{21} is the key attribute. E_3 has two single-valued attributes a_{31} and a_{32} of which a_{31} is the key attribute. The relationships do not have any attributes.

If a relational model is derived from the above ER model, then the minimum number of relations that would be generated if all the relations are in 3NF is _____.

Answer

For E_1 relation: $\langle \mathbf{a_{11}}, a_{12} \rangle$
For E_2 relation: $\langle \mathbf{a_{21}}, a_{22} \rangle$
For E_3 and R_{13} relationship: $\langle \mathbf{a_{31}}, a_{32}, \mathbf{a_{11}} \rangle$
For R_{12} : $\langle \mathbf{a_{11}}, \mathbf{a_{21}} \rangle$

Question

Aggregation is:
(A) an abstraction through which relationships are treated as lower level entities
(B) an abstraction through which relationships are treated as higher level entities
(C) an abstraction through which relationships are not treated at all as entities
(D) none of the above

Answer

Option B is the correct answer.

Question

Generalization is _____ process.

1.Top-down

2.Bottom up

3.Both (A) & (B)

4.None of these

Answer

Option 2 is the correct answer

Question

Let M and N be two entities in an E-R diagram with simple single value attributes R_1 and R_2 are two relationship between M and N, where as
R_1 is one-to-many and R_2 is many-to-many.
The minimum number of tables required to represent M, N, R_1 and R_2 in the relational model are_____.

Answer

There are a minimum of **3 tables required –**

- M
- $NR1$
- $R2$

Question

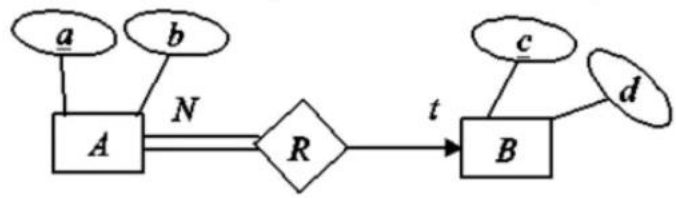
Consider the following statements S1 and S2 about the relational data model:
S1: A relation scheme can have at most one foreign key.
S2: A foreign key in a relation scheme R cannot be used to refer to tuples of R.
Which one of the following choices is correct?

Answer

We know that when we have multiple tables involved in a relation, we can have multiple foreign keys thus making **S1 false.** At the same time, there can be a case where the foreign and primary keys are the same. In such cases, we can use the foreign key to refer to tuples (rows) of R and hence even **S2 is false.**

Question

Find the minimum number of tables to realize this ER model



Answer

$$AR(\underline{a}, b, \underline{c})$$

$$B(\underline{c}, d)$$

Question

Which of the following is/are incorrect?

I. The strong Entity existence is dependent on any other Entity.

II. The strong Entity may or may not have the primary key.

III. The strong Entity is represented by a double rectangle.

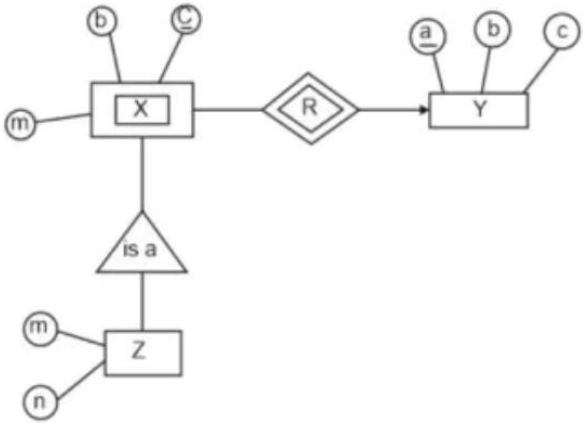
IV. The strong Entity indicates that each entity in a strong entity set can be uniquely identified.

Answer

Options I, II and III are incorrect.

Question

Find the minimum number of tables and number of foreign keys used to realize this ER model



Answer

For the top level, we can write –

$$XR(\underline{c}, m, b, \underline{a})$$

$$Y(\underline{a}, b, c)$$

Now, we can see that Z has been generalized into X. Hence, Z will have the primary key of X which will in turn contain the primary key of Y. Thus,

$$Z(m, n, \underline{c}, \underline{a})$$

Since Y is neither a subset and is also a strong entity, we try to check for its primary key which is $\underline{a}$. We can see that key $\underline{a}$ is used in both XR and Z. Hence, there are **3 tables and 2 foreign keys.**

Question

Consider the relation scheme R(E, F, G, H, I, J, K, L, M, N) and the set of functional dependencies-
{ E, F } → { G }
{ F } → { I , J }
{ E, H } → { K, L }
{ K } → { M }
{ L } → { N }

What is the key for R?
1.{ E, F }
2.{ E, F, H }
3.{ E, F, H, K, L }
4.{ E }

Also, determine the total number of candidate keys and super keys.

Answer

Option 2 is the correct answer. The number of super keys will be $2^7 = \mathbf{128}$

Question

Consider the given relation R with attributes F1,F2,F3,F4 and F5.
R(F1,F2,F3,F4,F5)
{
F1->F3
F3->F2
F2->F4
F4->F5
}

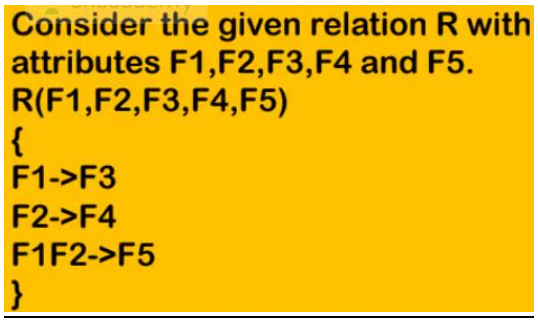
Find the candidate keys.

Answer

$$\{F1\} \rightarrow \{F1, F3, F2, F4, F5\}$$

Hence, $F1$ is the candidate key.

Question

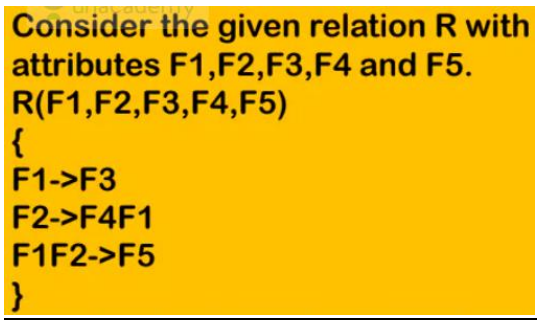


Find the candidate key

Answer

$$\{F1, F2\} \rightarrow \{F1, F2, F3, F4, F5\}$$

Question



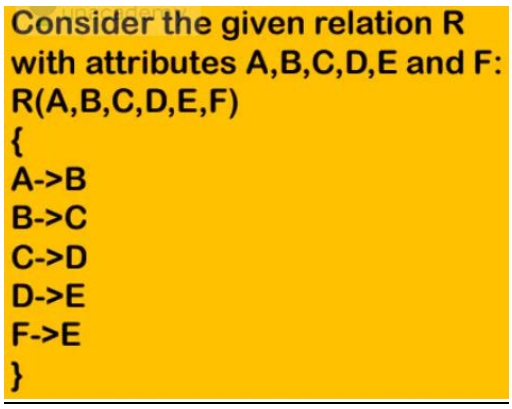
Find the candidate key

Answer

$$\{F2\} \rightarrow \{F2, F4, F1, F3, F5\}$$

Question

Find the candidate key.



Answer

$$\{A\} \rightarrow \{A, B, C, D, E\}$$

We can see that none of the keys derive F. Hence, the candidate key will be $\boldsymbol{AF}$

Question

Let a Relation R have attributes $\{a1, a2, a3 \dots an\}$ and the candidate key is $a1a2$. Then, find the number of possible super keys.

Answer

We know that,

$$No\ of\ super\ keys = 2^{(No\ of\ attributes - No\ of\ attributes\ in\ Candidate\ key)}$$

Now,

$$No\ of\ attributes = n$$

$$No\ of\ attributes\ in\ Candidate\ key = 2$$

Hence,

$$No\ of\ super\ keys = 2^{n-2}$$

Question

Let a Relation R have attributes {a1, a2, a3,…,an} and the candidate keys are "a1", "a2" then the possible number of super keys?

Answer

This is a different case since we have 2 separate candidate keys. Thus, in this case –

$$S(a1 \cup a2) = S(a1) + S(a2) - S(a1 \cap a2)$$

Where $S(a)$ denotes the number of super keys when a is the candidate key. Hence, we have –

$$S(a1 \cup a2) = 2^{n-1} + 2^{n-1} - 2^{n-2} = 2 * 2^{n-1} - 2^{n-2} = 2^n - 2^{n-2} = \mathbf{3 * 2^{n-2}}$$

Question

Let a Relation R have attributes {a1, a2, a3,…,an} and the candidate keys are "a1", "a2 a3" then the possible number of super keys?

Answer

$$S(a1 \cup a2a3) = S(a1) + S(a2a3) - S(a1 \cap a2a3) = 2^{n-1} + 2^{n-2} - 2^{n-3} = \mathbf{5 * 2^{n-3}}$$

Question

Let a Relation R have attributes {a1, a2, a3,…,an} and the candidate keys are "a1 a2", "a3 a4" then the possible number of super keys?

Answer

$$S(a1a2 \cup a3a4) = S(a1a2) + S(a3a4) - S(a1a2 \cap a3a4) = 2^{n-2} + 2^{n-2} - 2^{n-4} = \mathbf{7 * 2^{n-4}}$$

Question

Let a Relation R have attributes {a1, a2, a3,…,an} and the candidate keys are "a1 a2", "a1 a3" then the possible number of super keys?

Answer

$$S(a1a2 \cup a1a3) = S(a1a2) + S(a1a3) - S(a1a2 \cap a1a3) = 2^{n-2} + 2^{n-2} - 2^{n-3} = \mathbf{3 * 2^{n-3}}$$

Question

Let R (A, B, C, D, E, P, G) be a relational schema in which the following functional dependencies are known to hold:
AB → CD, DE → P, C → E, P → C and B → G. The relational schema R is
(A) in BCNF
(B) in 3NF, but not in BCNF
(C) in 2NF, but not in 3NF
(D) not in 2NF

GATE 2008

Answer

Here,

$$Candidate\ key = AB$$

Since we have a derivation as $B \rightarrow G$, we can conclude that the relation is **NOT in 2NF.**

Question

A table has fields F1, F2, F3, F4, F5 with the following functional dependencies
F1 → F3 , F2→ F4 , (F1 . F2) → F5
In terms of Normalization, this table is in
(A) 1 NF
(B) 2 NF
(C) 3 NF
(D) none

GATE 2005

Answer

$$Candidate\ key = F1F2$$

Since we have $F1 \rightarrow F3$, the relation is **NOT in 2NF form** and hence **Option 1** is the correct answer.

Question

Consider the schema R= (S, T, U, V) and the dependencies S→T, T→U, U→V and V→S. Let R (R1 and R2) be a decomposition such that R1∩R2 ≠ Ø. The decomposition is:
(A) Not in 2NF
(B) In 2NF but not in 3NF
(C) In 3NF but not in 2NF
(D) In both 2NF and 3NF

GATE 1999

Answer

$$Candidate\ key = S\ or\ T\ or\ U\ or\ V$$

Since all the attributes are prime attributes, the relation is **in BCNF form** and hence **Option D** is the correct answer.

Question

Consider the following four relational schemas. For each schema, all non-trivial functional dependencies are listed, The underlined attributes are the respective primary keys.

Schema I: ***Registration(rollno, courses)***
Field *'courses'* is a set-valued attribute containing the set of courses a student has registered for.
Non-trivial functional dependency
rollno → courses
Schema II: ***Registration (rollno, coursid, email)***
Non-trivial functional dependencies:
rollno, courseid → email
email → rollno
Schema III: ***Registration (rollno, courseid, marks, grade)***
Non-trivial functional dependencies:
rollno, courseid, → marks, grade
marks → grade
Schema IV: ***Registration (rollno, courseid, credit)***
Non-trivial functional dependencies:
rollno, courseid → credit
courseid → credit

GATE 2018

Which one of the relational schemas above is in 3NF but not in BCNF?
(A) Schema I
(B) Schema II
(C) Schema III
(D) Schema IV

Answer

Here,

- Schema IV is 1NF
- Schema III is 2NF

- Schema II is 3NF
- Schema I is BCNF

Hence, the correct answer is **Option B.**

Question

Consider a relation- R (A , B , C , D , E) with functional dependencies-

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

Find the highest NF form of this relation.

Answer

$$Candidate\ key = A, E, CD$$

Since we have $B \rightarrow D$ and B is **NOT a super key,** the above relation is in **3NF but no BCNF.**

Question

The relation scheme Student Performance (name, courseNo, rollNo, grade) has the following functional dependencies:
name, courseNo → grade
rollNo, courseNo → grade
name → rollNo
rollNo → name

Answer

$$Rel(N, C, R, G)$$

$$NC \rightarrow G$$

$$RC \rightarrow G$$

$$N \rightarrow R$$

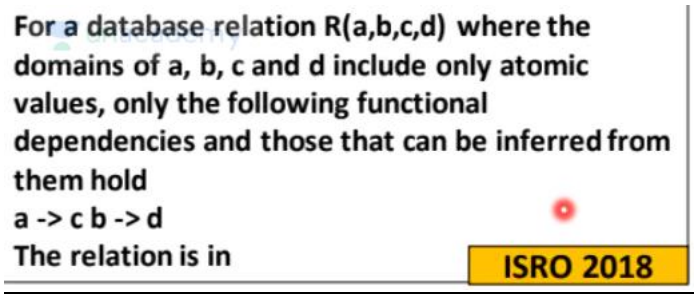
$$R \rightarrow N$$

Hence,

$$Candidate\ Key = NC, RC$$

The above relation is has $N \rightarrow R$ which is a partial dependency. Therefore, the table is **not in 2NF.**

Question

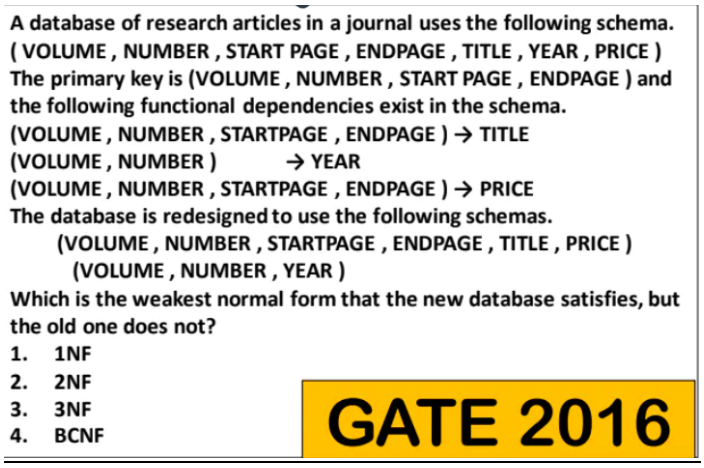


Answer

$$Candidate\ Key = ab$$

Since a NPA (c) is being derived from a partial candidate key (a), we can say that there is partial dependency and hence this is a **1NF form.**

Question



Answer

The old relation can be represented as –

$$R(V, N, S, E, T, Y, P)$$

$$VNSE \rightarrow T$$

$$VN \rightarrow Y$$

$$VNSE \rightarrow P$$

From the derivation $VN \rightarrow Y$, we can see that this has partial dependency and hence is a **1NF form.**

The new relation on the other hand can be represented as –

$$R(V, N, S, E, T, P)$$

$$VNSE \rightarrow T$$

$$VNSE \rightarrow P$$

Since there is no longer a partial dependency, we can conclude that the weakest NF which is satisfied by the new relation and not by the old relation is **2NF.**

Question

Which normal form is considered adequate for normal RDBMS?

Answer

3NF form

Question

Consider a relational table with a single record for each registered student with the following attributes.
1. Registration_Num: Unique registration number of each registered student
2. UID: Unique identity number, unique at the national level for each citizen
3. BankAccount_Num: Unique account number at the bank.
A student can have multiple accounts or join accounts. This attribute stores the primary account number.
4. Name: Name of the student
5. Hostel_Room: Room number of the hostel Which one of the following option is INCORRECT?

(A) BankAccount_Num is candidate key
(B) Registration_Num can be a primary key
(C) UID is candidate key if all students are from the same country
(D) If S is a superkey such that S∩UID is NULL then S∪UID is also a superkey

Answer

We know from the question that one student can have multiple bank accounts and 2 students can share a bank account (joint account). Hence, BankAccount_Num can't be a candidate key. So, **Option A is INCORRECT.**

Since it is mentioned that each student has a unique Registration_Num, it can be used as a Primary key for Student relation. Hence, **Option B is CORRECT.**

It is also mentioned that UID is a unique ID for people of the country. Hence, if all the students are of the same country, they will have unique UIDs and hence it can be used as a primary key in that scenario. Therefore, **Option C is CORRECT.**

Finally, let's assume Registration_Num is the Primary key. Then, we can have –

$$Superkey\ S = (Registration\_Num, Name)$$

$$S \cap UID = \phi$$

Now,

$$S \cup UID = (Registration\_Num, Name, UID) = S'$$

We can see that S' is also a super key. Therefore, **Option D is CORRECT.**

Question

The relation EMPDT1 is defined with attributes empcode(unique), name, street, city, state, and pincode. For any pincode,there is only one city and state. Also, for any given street, city and state, there is just one pincode. In normalization terms EMPDT1 is a relation in
(a) 1NF only
(b) 2NF and hence also in 1NF
(c) 3NF and hence also in 2NF and 1NF
(d) BCNF and hence also in 3NF, 2NF and 1NF

Answer

Here,

$$PA = \{empcode\}$$

$$NPA = \{name, street, city, state, pincode\}$$

From the question, we get –

$$street, city, state \rightarrow pincode$$

$$pincode \rightarrow street, city, state$$

We can see that there are no partial dependency, but there is a transitive dependency as a NPA is deriving other NPAs. Thus, the relation is in 2NF which also means it is in 1NF. Hence, **Option B** is the correct answer.

Question

Consider the following relational schemes for a library database:
Book (Title, Author, Catalog_no, Publisher, Year, Price)
Collection (Title, Author, Catalog_no)
With the following functional dependencies:
I. Title Author -> Catalog_no
II. Catalog_no -> Title Author Publisher Year
III. Publisher Title Year -> Price
Assume {Author, Title} is the key for both schemes.
Which of the following statements is true?
(a) Both Book and Collection are in BCNF
(b) Both Book and Collection are in 3NF only
(c) Book is in 2NF and Collection is in 3NF
(d) Both Book and Collection are in 2NF only

Answer

The relation Collection is in BCNF: Its given that {Author, Title} is the key and there is only one functional dependency (FD) applicable to the relation Collection {i.e. Title Author –> Catalog_no}.
As per the definitions of the normal forms Book is in 2NF.

Hence, **Option C** is the correct answer.

Question

The best normal form of relation scheme R(A, B, C, D) along with the set of functional dependencies F = {AB → C, AB → D, C → A, D → B} is

Answer

$$PA = \{A, B, C, D\}$$

Hence, the relation is **3NF.** It is not BCNF since $C \rightarrow A$ has LHS which is not a super key.

Question

Which of the following is false?
(A) Every binary relation is never be in BCNF.
(B) Every BCNF relation is in 3NF.
(C) 1 NF, 2 NF, 3 NF and BCNF are based on functional dependencies.
(D) Multivalued Dependency (MVD) is a special case of Join Dependency (JD).

Answer

None of the options are FALSE statements

Question

Consider the relation $R(A, B, C, D)$ with the following dependencies –

$$AB \rightarrow C$$

$$D \rightarrow A$$

Suppose we decompose the relation into $R1(A, D)$ and $R2(B, C, D)$. Is the decomposition dependency preserving?

Answer

In this case, we can get $D \rightarrow A$ using $R1$ but under no circumstances will we be able to achieve $AB \rightarrow C$. Hence, this is a **non – dependency preserving** decomposition.

Question

Consider the relation $R(A, B, C, D, E)$ with the following dependencies –

$$AB \rightarrow CD$$

$$C \rightarrow D$$

$$D \rightarrow E$$

Suppose we decompose the relation into

- $R1(A, B, C)$

- $R2(C, D)$
- $R3(D, E)$

Is the decomposition dependency preserving?

Answer

From $R2$ and $R3$, we can directly get $C \rightarrow D$ and $D \rightarrow E$. Now, from $R1$, we get $AB \rightarrow C$ and from $R2$ we get $C \rightarrow D$. Thus, we can also get $AB \rightarrow CD$. Therefore, this is **dependency preserving** decomposition.

Question

Consider the following relations:

Student

Roll_No	Student_Name
1	Raj
2	Rohit
3	Raj

Performance

Roll_No	Course	Marks
1	Math	80
1	English	70
2	Math	75
3	English	80
2	Physics	65
3	Math	80

Consider the following SQL query.
SELECT S. Student_Name, sum (P.Marks)
FROM Student S, Performance P
WHERE S. Roll_No =P.Roll_No
GROUP BY S.Student_Name
The number of rows that will be returned by the SQL query is _____.

Answer

Student_Name	Marks
Raj	310
Rohit	140

One thing to note here is that in the Student table, the name **Raj** appears twice. Since the query is grouping by Student name, the result will have only 2 tuples.

Question

Consider a database that has the relation schema EMP (EmpId, EmpName, and DeptName). An instance of the schema EMP and a SQL query on it are given below:

EMP

EmpId	EmpName	DeptName
1	XYA	AA
2	XYB	AA
3	XYC	AA
4	XYD	AA
5	XYE	AB
6	XYF	AB
7	XYG	AB
8	XYH	AC
9	XYI	AC
10	XYJ	AC

```
SELECT AVG(EC.Num)
FROM EC
WHERE (DeptName, Num) IN
    (SELECT DeptName, COUNT(EmpId) AS
                        EC( DeptName, Num)
    FROM EMP
    GROUP BY DeptName)
```

The output of executing the SQL query is_____.

Answer

First, we will execute the sub – query

```
SELECT DeptName, COUNT(EmpID) AS EC(DeptName,Num) FROM EMP GROUP BY DeptName
```

This will result in the following table –

DeptName	Num
AA	4
AB	3
AC	3
AD	2
AE	1

EC

Now, we can execute the original query and we get –

$$AVG = \mathbf{2.6}$$

Question

Consider the following database table named *top_scorer*.

top_scorer

player	country	goals
Klose	Germany	16
Ronaldo	Brazil	15
G Muller	Germany	14
Fontaine	France	13
Pele	Brazil	12
Klinsmann	Germany	11
Kocsis	Hungary	11
Batistuta	Argentina	10
Cubillas	Peru	10
Lato	Poland	10
Lineker	England	10
T Muller	Germany	10
Rahn	Germany	10

Consider the following SQL query:

```
SELECT ta.player FROM top_scorer AS ta
WHERE ta.goals > ALL (SELECT tb.goals
                      FROM top_scorer AS tb
                      WHERE tb.country = 'Spain')
AND ta.goals > ANY (SELECT tc.goals
                    FROM top_scorer AS tc
                    WHERE tc.country = 'Germany')
```

The number of tuples returned by the above SQL query is _____

Answer

Here we have 2 sub – queries as follows –

- `SELECT tb.goals FROM top_scorer AS tb WHERE tb.country = 'Spain'`
- `SELECT tc.goals FROM top_scorer AS tc WHERE tc.country = 'Germany'`

Solving, we get that **7 tuples** are returned.

Question

water schemes		
scheme_no	district_name	capacity
1	Ajmer	20
1	Bikaner	10
2	Bikaner	10
3	Bikaner	20
1	Churu	10
2	Churu	20
1	Dungargarh	10

the number of tuples returned by the following SQL query is:

```
with total (name, capacity) as
    select district_name, sum (capacity)
    from water_schemes
    group by district_name
with total_avg (capacity) as
    select avg (capacity)
    from total
select name
    from total, total_avg
    where total.capacity ≥ total_avg. capacity
```

Answer

The **WITH** clause is used to form temporary relations. Here, we create 2 temporary relations –

- Total
- Total_avg

The query will return **2 tuples –** (Bikaner, Churu)

Question

Database table by name Loan_Records is given below:

Borrower	Bank_Manager	Loan_Amount
Ramesh	Sunderajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00

What is the output of the following SQL query?

```
SELECT count(*)
FROM (
    (SELECT Borrower, Bank_Manager FROM
    Loan_Records) AS S NATURAL JOIN
    (SELECT Bank-Manager, Loan_Amount
    FROM Loan_Records) AS T
    );
```

Answer

Firstly, inner queries get executed

(SELECT Borrower, Bank-Manager Loan_Records) AS S

S

Borrower	Bank-Manager
Ramesh	Sunderajan
Suresh	Ramgopal
Mahesh	Sunderajan

(SELECT Bank-Manager, Loan_Amount FROM Loan_Records) AS T

T

Bank-Manager	Loan_Amount
Sunderajan	10000.00
Ramgopal	5000.00
Sunderajan	7000.00

S Natural Join T

Borrower	Bank-Manager	Loan_Amount
Ramesh	Sunderajan	10000.00
Ramesh	Sunderajan	7000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00
Mahesh	Sunderajan	10000.00

Hence, the SQL query will return **5**

Question

Which of the following statements are TRUE about an SQL query?

P : An SQL query can contain a HAVING clause even if it does not have a GROUP BY clause

Q : An SQL query can contain a HAVING clause only if it has a GROUP BY clause

R : All attributes used in the GROUP BY clause must appear in the SELECT clause

S : Not all attributes used in the GROUP BY clause need to appear in the SELECT clause

(a) P and R (b) P and S

(c) Q and R (d) Q and S

Answer

In a SQL query, if we use HAVING without a GROUP BY clause, then the HAVING clause acts as a WHERE clause. Hence, **Statement P is True but Statement Q is False**. Additionally, there can be queries where we don't select an attribute but refer to that attribute in GROUP BY. Hence, **Statement R is False but Statement S is True.** Hence, **Option B** is the correct answer.

Question

Consider the following two tables and four queries in SQL.

Book (isbn, bname), Stock (isbn, copies)Query 1:

SELECT B.isbn, S.copies FROM Book B INNER JOIN Stock S ON B.isbn = S.isbn; Query 2:

SELECT B.isbn, S.copies FROM B B LEFT OUTER JOIN Stock S ON B.isbn = S.isbn; Query 3:

SELECT B.isbn, S.copies FROM Book B RIGHT OUTER JOIN Stock S ON B.isbn = S.isbn; Query 4:

SELECT B.isbn, S.copies FROM B B FULL OUTER JOIN Stock S ON B.isbn = S.isbn;

Which one of the queries above is certain to have an output that is a superset of the outputs of the other three queries?

Answer

We know that FULL OUTER JOIN will have the maximum coverage out of all the other options. Hence, **Query 4** is the correct answer

Question

Consider the following relation

Cinema (theater, address, capacity) Which of the following options will be needed at the end of the SQL query

SELECT P1. address FROM Cinema P1 Such that it always finds the addresses of theaters with maximum capacity?

(A) WHERE P1. Capacity> = All (select P2. Capacity from Cinema P2)

(B) WHERE P1. Capacity> = Any (select P2. Capacity from Cinema P2)

(C) WHERE P1. Capacity > All (select max(P2. Capacity) from Cinema P2)

(D) WHERE P1. Capacity > Any (select max (P2. Capacity) from Cinema P2)

Answer

Basically, we need the query to be such that P1.Capacity is greater than or equal to **ALL** of the other cinema capacities. Hence **Option A** is the correct answer.

Question

A relational schema for a train reservation database is given below.
Passenger (pid, pname, age)
Reservation (pid, class, tid)

What pids are returned by the following SQL query for the above instance of the tables?
SELECT pid FROM Reservation , WHERE class 'AC' AND EXISTS (SELECT * FROM Passenger WHERE age > 65 AND Passenger. pid = Reservation.pid)

(A) 1, 0
(B) 1, 2
(C) 1, 3
(D) 1, 5

pid	pname	age
0	Sachin	65
1	Rahul	66
2	Sourav	67
3	Anil	69

Table : Reservation

pid	class	tid
0	AC	8200
1	AC	8201
2	SC	8201
5	AC	8203
1	SC	8204
3	AC	8202

Answer

Option C

Question

Database table by name overtime_allowance is given below

ISRO 2017

Employee	Department	OT_allowance
RAMA	Mechanical	5000
GOPI	Electrical	2000
SINDHU	Computer	4000
MAHESH	Civil	1500

What is the output of the following SQL query?

```
select count(*) from ((select Employee, Department from Overtime_allowance) as S
natural join (select Department, OT_allowance from Overtime_allowance) as T);
```

Answer

Common attributes in both the table column are the department. So, we apply natural join, it will give the output as common tuples in both the table S and R.

Question

SID	SNAME	RATING	AGE
22	Dustin	7	45
29	Borg	1	33
31	Pathy	8	55
32	Robert	8	25
58	Raghu	10	17
64	Herald	7	35
71	Vishnu	10	16
74	King	9	35
85	Archer	3	26
84	Bob	3	64
96	Flinch	3	17

For the query
SELECT S.rating, AVG(S.age) AS average FROM Sailors S
Where S.age >= 18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT(*) FROM Sailors S2 where S.rating = S2.rating)

The number of rows returned is
(A) 6
(B) 5
(C) 4
(D) 3

Answer

Option D

Question

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Π NAME, CITY (CUSTOMER)

Answer

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

Question

$$\pi_{Id,\ Name}\left(\sigma_{Hobby='stamps'\ OR\ Hobby='coins'}(Person)\right)$$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Find the number of rows returned.

Answer

The query will return 2 rows –

- 1123, John
- 9876, Bart

Question

Write the relational algebra query for the following SQL query –

```
SELECT C.CrsName
FROM Course C, Teaching T
WHERE C.CrsCode=T.CrsCode AND T.Semester='S2000'
```

Answer

$$\pi_{C.CrsName}\left(\sigma_{C.CrsCode=T.CrsCode\ \wedge\ T.Semester='S2000'}(C \times T)\right)$$

We can also write –

$$\pi_{CrsName}(\text{Course} \bowtie \sigma_{Semester='S2000'}(\text{Teaching}))$$
$$\pi_{CrsName}(\sigma_{Sem='S2000'}(\text{Course} \bowtie \text{Teaching}))$$

Question

User

Id	Name	Age	Gender	OccupationId	CityId
1	John	25	Male	1	3
2	Sara	20	Female	3	4
3	Victor	31	Male	2	5
4	Jane	27	Female	1	3

Occupation

OccupationId	OccupationName
1	Software Engineer
2	Accountant
3	Pharmacist
4	Library Assistant

City

CityId	CityName
1	Halifax
2	Calgary
3	Boston
4	New York
5	Toronto

Find the output for $User \bowtie Occupation \bowtie City$

Answer

CityId	OccupationId	Id	Name	Age	Gender	OccupationName	CityName
3	1	1	John	25	Male	Software Engineer	Boston
4	3	2	Sara	20	Female	Pharmacist	New York
5	2	3	Victor	31	Male	Accountant	Toronto
3	1	4	Jane	27	Female	Software Engineer	Boston

Question

Consider the relational schema given below, where **eId** of the **dependent** is a foreign key referring to **empId** of the relation **employee**. Assume that every employee has at least one associated dependent in the **dependent** relation.

employee (empId, empName, empAge)
dependent(depId, eId, depName, depAge)

Consider the following relational algebra query:

$\Pi_{empId}(employee) - \Pi_{empId}(employee \bowtie_{(empId = eId) \wedge (empAge \leq depAge)} dependent)$

The above query evaluates to the set of empIds of employees whose age is greater than that of
(a) some dependent.
(b) all dependents.
(c) some of his/her dependents.
(d) all of his/her dependents.

Answer

The above relational algebra query will return the **difference** of the following sets –

- All empIDs
- All empIDs of the employees who are younger than or the same age as their dependents

Hence, **Option D** is the correct answer

Question

Consider two relations R_1 (A, B) with the tuples (1, 5), (3, 7) and R_2(A, C) = (1, 7), (4, 9).

Assume that R(A,B,C) is the full natural outer join of R_1 and R_2. Consider the following tuples of the form (A,B,C): a = (1, 5, null), b = (1, null, 7), c = (3, null, 9), d = (4, 7, null), e = (1, 5, 7), f = (3, 7, null), g = (4, null, 9). Which one of the following statements is correct?

(a) R contains a, b, e, f, g but not c, d
(b) R contains all of a, b, c, d, e, f, g
(c) R contains e, f, g but not a, b
(d) R contains e but not f, g

Answer

R_1

A	B
1	5
3	7

R_2

A	C
1	7
4	9

$R = R_1 \bowtie R_2$

R	A	B	C	
	1	5	7	e
	3	7	Null	f
	4	Null	9	g

Hence, **Option C** is correct.

Question

Consider the join of a relation R with a relation S. If R has m tuples and S has n tuples then the maximum and minimum sizes of the join respectively are

(a) m + n and 0
(b) mn and 0
(c) m + n and $|m - n|$
(d) mn and m + n

Answer

Option B is correct answer

Question

With regard to the expressive power of the formal relational query languages, which of the following statements is true?

(a) Relational algebra is more powerful than relational calculus
(b) Relational algebra has the same power as relational calculus
(c) Relational algebra has the same power as safe relational calculus
(d) None of the above

Answer

(c)

Every query that can be expressed using a safe relational calculus query can also be expressed as a relational algebra query.
Therefore, relational algebra has the same power as safe relational calculus.

Question

Let r be a relation instance with schema R = (A, B, C, D). We define $r_1 = \Pi_{A,B,C}(r)$ and $r_2 = \Pi_{A,D}(r)$. let $S = r_1 * r_2$ where * denotes natural join. Given that the decomposition of r into r_1 and r_2 is lossy, which one of the following is TRUE?

(a) $s \subset r$
(b) $r \cup s = r$
(c) $r \subset s$
(d) $r * s = s$

[2005 : 1 Mark]

Answer

We know that, if a relation R is decomposed into $R1$ and $R2$, then for a **lossy decomposition** –

$$R \subset R1 \bowtie R2$$

Since it is given that $S = r1 * r2$ where $*\rightarrow\bowtie$, we can write –

$$r \subset s$$

Hence, **Option C** is the correct answer.

Question

Which of the following relational query languages have the same expressive power?

I. Relational algebra.

II. Tuple relational calculus restricted to safe expressions.

III. Domain relational calculus restricted to safe expressions.

(a) II and III only (b) I and II only

(c) I and III only (d) I, II and III

Answer

Option D

Question

Consider a database that has the relation schema CR(StudentName, CourseName). An instance of the schema CR is as given below:

CR

Student Name	Course Name
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database.

$T_1 \leftarrow \pi_{CourseName}(\sigma_{StudentName = 'SA'}(CR))$

$T_2 \leftarrow CR \div T_1$

The number of rows in T_2 is _____.

Answer

The result of $T1$ will be – $\{CA, CB, CC\}$

So, the result of $T2$ will be the students who study all the three courses in T1. Hence, we can see that $T2 = \{SA, SC, SD, SF\}$. Hence, the number of rows will be **4.**

Question

Consider the relations r(A, B) and s(B, C), where s.B is a primary key and r.B is a foreign key referencing s.B. Consider the query
Q: $r \bowtie (\sigma_{B<5}(s))$
Let LOJ denote the natural left outer-join operation. Assume that r and s contain no null values.
Which one of the following is NOT equivalent to Q? GATE CS 2018
(A) $\sigma_{B<5}(r \bowtie s)$
(B) $\sigma_{B<5}(r \text{ LOJ } s)$
(C) $r \text{ LOJ } (\sigma_{B<5}(s))$
(D) $\sigma_{B<5}(r) \text{LOJ } s$

Answer

Option C is correct answer. To solve this, assume the following –

A	B
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

R

A	B
1	2
2	4
3	6
4	8
5	10
6	12
7	14

S

A	B	C
1	1	2
2	2	4
3	3	6
4	4	8

Q

Question

Consider the following relations P(X, Y. Z), Q(X, Y, T) and R(Y, V).

P		
X	Y	Z
X1	Y1	Z1
X1	Y1	Z2
X2	Y2	Z2
X2	Y4	Z4

Q		
X	Y	T
X2	Y1	2
X1	Y2	5
X1	Y1	6
X3	Y3	1

R	
Y	V
Y1	V1
Y3	V2
Y2	V3
Y2	V2

How many tuples will be returned by the following relational algebra query?
$\Pi_x(\sigma_{(P.Y = R.Y \wedge R.V = V2)}(P \times R)) - \Pi_x(\sigma_{(Q.Y = R.Y \wedge Q.T > 2)}(Q \times R))$

Answer

The query returns **one tuple.**

Question

Let R and S be two relations with the following schema
R (P,Q,R1,R2,R3)
S (P,Q,S1,S2)
Where {P, Q} is the key for both schemas. Which of the following queries are equivalent?

GATE 2007

(A) Only I and II
(B) Only I and III
(C) Only I, II and III
(D) Only I, III and IV

I. $\Pi_P (R \bowtie S)$
II. $\Pi_P (R) \bowtie \Pi_P (S)$
III. $\Pi_P (\Pi_{P,Q} (R) \cap \Pi_{P,Q} (S))$
IV. $\Pi_P (\Pi_{P,Q} (R) - (\Pi_{P,Q} (R) - \Pi_{P,Q} (S)))$

Answer

Option D is correct answer

Question

Which one of the following is NOT a part of the ACID properties of database transactions?

(a) Atomicity (b) Consistency
(c) Isolation (d) Deadlock-freedom

[2016 (Set-1) : 1 Mark]

Answer

Option D is the correct answer

Question

Amongst the ACID properties of a transaction, the 'Durability' property requires that the changes made to the database by a successful transaction persist

A
Except in case of an Operating System crash
B
Except in case of a Disk crash
C
Except in case of a power failure
D
Always, even if there is a failure of any kind

GATE 2005

Answer

Option D is the correct answer

Question

Which Type Of Schedule ?

1. Recoverable Schedule
2. Irrecoverable Schedule
3. Both
4. None

Transaction T1	Transaction T2
R(A)	
W(A)	
	R(A) // Dirty Read
	W(A)
Commit	
	Commit // Delayed

Answer

Option 1.

Question

Which Type Of Schedule ?

1. Cascadless Schedule
2. Cascading Schedule
3. Strict Schedule
4. Irrecoverable Schedule

T1	T2	T3
R(A)		
W(A)		
	R(A)	
	W(A)	
		R(A)
		W(A)
Failure		

Answer

Option 2

Question

Which Type Of Schedule ?

1. Cascadless Schedule
2. Cascading Schedule
3. Strict Schedule
4. Irrecoverable Schedule

T1	T2
R(A)	
W(A)	
Commit	
	R(A)
	W(A)
	Commit

Answer

Option 1

Question

Which Type Of Schedule ?

1. Cascadless Schedule
2. Cascading Schedule
3. Strict Schedule
4. Irrecoverable Schedule

T1	T2
W(A)	
Commit/ Rollback	
	R(A)/W(A)

Answer

Option 3

Question

T_1	T_2
R(A)	
	R(A)
W(A)	
commit	
	W(A)
	R(A)
	commit

Is the above schedule a strict schedule?

Answer

Yes, as it supports committed write operations.

Question

Which of the following scenarios may lead to an irrecoverable error in a database system?

A.A transaction writes a data item after it is read by an uncommitted transaction.
B.A transaction reads a data item after it is read by an uncommitted transaction.
C.A transaction reads a data item after it is written by a committed transaction.
D.A transaction reads a data item after it is written by an uncommitted transaction.

Answer

Option D

Question

T_1	T_2
R(A)	
W(A)	
	W(A)
	R(A)
	commit
abort	

Is the above schedule recoverable or non – recoverable?

Answer

T2 reads the value of A and then commits before T1. Later, T1 aborts and hence causing data inconsistency. Therefore, the schedule is **non – recoverable.**

Question

T_1	T_2	T_3
R(A)		
	W(A)	
	Commit	
W(A)		
		W(A)
		Commit
Commit		

Which of the following is true?
(A) The schedule is view serializable schedule and strict recoverable schedule
(B) The schedule is non-serializable schedule and strict recoverable schedule
(C) The schedule is non-serializable schedule and is not strict recoverable schedule.
(D) The Schedule is serializable schedule and is not strict recoverable schedule

Answer

write – write pair done by transactions T_1 followed by T_3 which is violating the above-mentioned condition of strict schedules as T_3 is supposed to do write operation only after T_1 commits which is violated in the given schedule. Hence the given schedule is serializable but not strict recoverable.
So, option (D) is correct.

Question

Choose the correct option about the following schedule.

S: R2(A); W3(A); (commit T3); W1(A); (commit T1); W2(B); R2(C); (commit T2); R4(A); R4(B); (commit T4)

A. S is both recoverable and conflict serializable
B. S is neither recoverable nor conflict serializable
C. S is recoverable but not conflict serializable
D. S is not recoverable but conflict serializable

Answer

Option C is the correct answer

Question

All strict schedules are and all cascadeless schedules are .
A. Cascadeless, Recoverable
B. Serial, Recoverable
C. Recoverable, Conflict serializable
D. None of the choices

Answer

Option A

Question

Consider the following database schedule with two transactions, T1 and T2.

`S = r2(X); r1(X); r2(Y); w1(X); r1(Y); w2(X); a1; a2;`

where ri(Z) denotes a read operation by transaction Ti on a variable Z, wi(Z) denotes a write operation by Ti on a variable Z and ai denotes an abort by transaction Ti .

Which one of the following statements about the above schedule is TRUE?

(A) S is non-recoverable

(B) S is recoverable, but has a cascading abort

(C) S does not have a cascading abort

(D) S is strict

GATE 2016

Answer

T1	T2
	R(X)
R(X)	
	R(Y)
W(X)	
R(Y)	
	W(X)
Abort	
	Abort

There is no dirty read in this case and hence S is recoverable. Thus, **Option A is FALSE.** Additionally, since there is no commit statements, S will not be strict. Hence, **Option D is also FALSE.** Now, when

T2 aborts before T1, then we would have to cascade the abort process. However, since T1 aborts before T2, there will be no cascade abort. Hence, **Option B is FALSE and Option C is TRUE.**

Question

Identify the number of correct statements.

1. A transaction is a set of instructions that performs a logical unit of work.

2. A transaction can be executed partially.

3. A transaction can have one/more database access operations like insertion deletion, modification or retrieval operations.

4. In order to ensure database remains in stable state after the transaction is executed, a transaction must ensure atleast one among the four ACID properties.

Number of correct statements _____.

Answer

- A transaction is a set of instructions/operations that perform a logical unit of work.
- A transaction can't be executed partially.
- A transaction can perform insertion, deletion, modification (write) and retrieval (read).
- A transaction needs to satisfy all 4 ACID properties.

Hence, only **Statement 1 is TRUE.**

Question

Which of the following statements are true?

i) Changes in data done by execution of an instruction of a transaction are directly stored in the database.

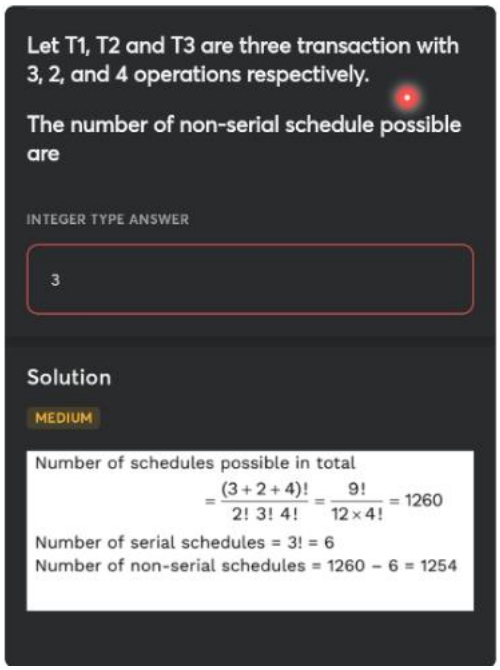
ii) Database in committed & aborted state of the transaction is always consistent.

iii) Phantom read problem during concurrency of transactions is about not finding a variable.

iv) Normalization is the solution to the redundancy problem.

Answer

Statements 2, 3 and 4 are correct.



Question

Find out if the below schedule is conflict serializable or not.

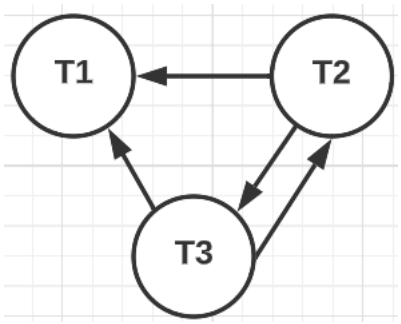
T_1	T_2	T_3
	R(I)	
		R(K)
		W(I)
	W(I)	
	W(J)	
		W(K)
R(I)		
R(J)		
W(I)		
W(J)		

Answer

The conflicting operations are –

- $R2(I) \rightarrow W3(I)$
- $W3(I) \rightarrow W2(I)$
- $W3(I) \rightarrow W1(I)$
- $W3(I) \rightarrow R1(I)$
- $W2(I) \rightarrow R1(I)$
- $W2(I) \rightarrow W1(I)$
- $W2(J) \rightarrow R1(J)$
- $W2(J) \rightarrow W1(J)$

Thus, we can draw the precedence graph as follows –



Since there is a cycle, the schedule is **non – conflict serializable.**

Question

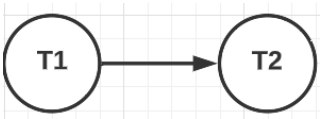
S1 :

T_1	T_2
R(P)	
W(Q)	
	R(P)
	W(P)
R(Q)	
W(Q)	
	R(Q)
	W(Q)

Conflict serializable?

Answer

Precedence graph is as follows –



As there are no cycles, the schedule is **conflict serializable.**

Question

S2 :

T_1	T_2
R(P)	
W(Q)	
	R(P)
	W(P)
	R(Q)
	W(Q)
R(Q)	
W(Q)	

Answer

Not conflict serializable

Question

T_1	T_2
R(P)	
R(Q)	
	R(P)
	R(Q)
	W(Q)
W(Q)	

Answer

Not conflict serializable

Question

T_1	T_2
R(P)	
	R(P)
	R(Q)
	W(Q)
R(Q)	
W(P)	

Answer

Conflict serializable

Question

5.11 Consider two transactions T_1 and T_2, and four schedules S_1, S_2, S_3, S_4 of T_1 and T_2 as given below:

$T_1 : R_1[x]\ W_1[x]\ W_1[y]$

$T_2 : R_2[x]\ R_2[y]\ W_2[y]$

$S_1 : R_1[x]\ R_2[x]\ R_2[y]\ W_1[x]\ W_1[y]\ W_2[y]$

$S_2 : R_1[x]\ R_2[x]\ R_2[y]\ W_1[x]\ W_2[y]\ W_1[y]$

$S_3 : R_1[x]\ W_1[x]\ R_2[x]\ W_1[y]\ R_2[y]\ W_2[y]$

$S_4 : R_2[x]\ R_2[y]\ R_1[x]\ W_1[x]\ W_1[y]\ W_2[y]$

Which of the above schedules are conflict-serializable?

(a) S_1 and S_2 (b) S_2 and S_3

(c) S_3 only (d) S_4 only

[2009 : 2 Marks]

Answer

Option B

Question

5.13 Consider the following schedule for transactions T1, T2 and T3:

T1	T2	T3
Read (X)		
	Read (Y)	
		Read (Y)
	Write (Y)	
Write (X)		
		Write (X)
	Read (X)	
	Write (X)	

Which one of the schedules below is the correct serialization of the above?

(a) T1 → T3 → T2

(b) T2 → T1 → T3

(c) T2 → T3 → T1

(d) T3 → T1 → T2

[2010 : 2 Marks]

Answer

Option A

Question

5.14 Consider the following transactions with data items P and Q initialized to zero:

T_1 : read (P);
read (Q) ;
if P = 0 then Q: = Q + 1;
write (Q);

T_2 : read (Q);
read (P);
if Q = 0 then P: = P + 1;
write (P);

Any non-serial interleaving of T_1 and T_2 for concurrent execution leads to

(a) a serializable schedule

(b) a schedule that is not conflict serializable

(c) a conflict serializable schedule

(d) a schedule for which a precedence graph cannot be drawn

[2012 : 2 Marks]

Answer

Option B

Question

5.16 Consider the following four schedules due to three transactions (indicated by the subscript) using *read* and *write* on a data item x, denoted by $r(x)$ and $w(x)$ respectively. Which one of them is conflict serializable?

(a) $r_1(x); r_2(x); w_1(x); r_3(x); w_2(x)$
(b) $r_2(x); r_1(x); w_2(x); r_3(x); w_1(x)$
(c) $r_3(x); r_2(x); r_1(x); w_2(x); w_1(x)$
(d) $r_2(x); w_2(x); r_3(x); r_1(x); w_1(x)$

[2014 (Set-1) : 2 Marks]

Answer

Option D

Question

5.17 Consider the following schedule S of transactions T1, T2, T3 and T4:

T1	T2	T3	T4
	Reads(X)		
		Writes(X)	
Writes(X)		Commit	
Commit			
	Writes(Y)		
	Reads(Z)		
	Commit		
			Reads(X)
			Reads(Y)
			Commit

Which one of the following statements is CORRECT?

(a) S is conflict-serializable but not recoverable
(b) S is not conflict-serializable but is recoverable
(c) S is both conflict-serializable and recoverable
(d) S is neither conflict-serializable nor is it recoverable

[2014 (Set-2) : 2 Marks]

Answer

Option C

Question

5.18 Consider the transactions $T1$, $T2$, and $T3$ and the schedules $S1$ and $S2$ given below.

$T1$: $r1\ (X)$; $r1\ (Z)$; $w1\ (X)$; $w1(Z)$
$T2$: $r2\ (Y)$; $r2\ (Z)$; $w2\ (Z)$
$T3$: $r3\ (Y)$; $r3\ (X)$; $w3\ (Y)$

S1 : $r1\ (X);\ r3\ (Y);\ r3\ (X);\ r2\ (Y);\ r2\ (Z);\ w3(Y);\ w2\ (Z);\ r1(Z);\ w1(X);\ w1(Z)$

S2 : $r1\ (X);\ r3\ (Y);\ r2\ (Y);\ r3\ (X);\ r1(Z);\ r2(Z);\ w3\ (Y);\ w1\ (X);\ w2\ (Z);\ w1(Z)$

Which one of the following statements about the schedules is **TRUE**?

(a) Only S1 is conflict-serializable.
(b) Only S2 is conflict-serializable.
(c) Both S1 and S2 are conflict-serializable.
(d) Neither S1 nor S2 is conflict-serializable.

Answer

Option A

Question

Let $R_i(z)$ and $W_i(z)$ denote reac and write operations on a data element z by a transaction T_i, respectively. Consider the schedule S with four transactions.

S : $R_4(x), R_2(x), R_3(x), R_1(y), W_1(y), W_2(x), W_3(y), R_4(y)$

Which one of the following serial schedules is conflict equivalent to S?

A $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$

B $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$

C $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

D $T_3 \rightarrow T_1 \rightarrow T_4 \rightarrow T_2$

Answer

Option A

Question

Let $r_i(z)$ and $w_i(z)$ denote read and write operations respectively on a data item z by a transaction T_i. Consider the following two schedules.

$S_1 : r_1(x)\ r_1(y)\ r_2(x)\ r_2(y)\ w_2(y)\ w_1(x)$

$S_2 : r_1(x)\ r_2(x)\ r_2(y)\ w_2(y)\ r_1(y)\ w_1(x)$

Which one of the following options is correct?

A S_1 is conflict serializable, and S_2 is not conflict serializable.

B S_1 is not conflict serializable, and S_2 is conflict serializable.

C Both S_1 and S_2 are conflict serializable.

D Neither S_1 nor S_2 is conflict serializable.

Answer

Option B

Question

Consider a schedule of transactions T_1 and T_2 :

T_1	RA			RC		WD		WB	Commit	
T_2		RB	WB		RD		WC			Commit

Here, RX stands for "Read(X)" and WX stands for "Write(X)". Which one of the following schedules is conflict equivalent to the above schedule?

A

T_1				RA	RC	WD	WB		Commit	
T_2	RB	WB	RD					WC		Commit

B

T_1	RA	RC	WD	WB					Commit	
T_2					RB	WB	RD	WC		Commit

C

T_1	RA	RC	WD				WB		Commit	
T_2				RB	WB	RD		WC		Commit

D

T_1					RA	RC	WD	WB	Commit	
T_2	RB	WB	RD	WC						Commit

Answer

Option A

Question

Consider the following partial Schedule S involving two transactions $T1$ and $T2$. Only the read and the write operations have been shown. The read operation on data item P is denoted by read (P) and the write operation on data item P is denoted by write (P).

Time instance	Transaction-id	
	T1	*T2*
1	*read(A)*	
2	*write(A)*	
3		*read(C)*
4		*write(C)*
5		*read(B)*
6		*write(B)*
7		*read(A)*
8		*commit*
9	*read(B)*	

Schedule S

Suppose that the transaction $T1$ fails immediately after time instance 9. Which one of the following statements is correct?

A $T2$ must be aborted and then both $T1$ and $T2$ must be re-started to ensure transaction atomicity

B Schedule S is non-recoverable and cannot ensure transaction atomicity

C Only $T2$ must be aborted and then re-started to ensure transaction atomicity

D Schedule S is recoverable and can ensure atomicity and nothing else needs to be done

Answer

Option B

Question

A non serializable schedule is a concurrent schedule. whose result is not equal to any serial schedule.

For a schedule S, having transactions T_1 and T_2 as follows:

$T_1 : w_1(a)\ w_1(b)\ r_1(a)$

$T_2 : r_2(b)\ r_2(a)$

Number of non serializable schedules possible _____.

Answer

Basically, we need to find the possible combinations of the operations that cause cycle from $T1 \rightarrow T2$ and also from $T2 \rightarrow T1$. This can be done by trial and error. The correct answer is **4.**

Question

Consider the following relational data.

R

A	B	C
2	3	6
5	1	2
3	1	7

S

C	D
6	3
2	5
8	1

For the following operations ,R S and R × S, which one of the following statements are true. where '⋈' is Natural Join and 'X' is cross product

(1)Number of tuples returned by R × S and R ⋈ S are 9

(2)Attribute set of R × S is (A, B, C, C, D) and that of R ⋈ S is (A, B, C, D)

(3)Number of tuples returned of R × S are 9 and R ⋈ S are 2

(4)Attribute set of R × S is (A, B, C, C, D) and that of R ⋈ S is (A, B, C, C, D)

No. of correct statements _____.

Answer

Statement (2) and (3) are correct.

Question

Assume that there are two transactions T1 and T2 , of which T1 started execution before T2. Consider the following schedule involving two transactions T1 and T2.
T1 : R(X) , T2 : R(Y) , T2 : W(X) , T1 : W(X)

Which o the following is TRUE about the above schedule ?
A) It is allowed under Basic time stamp order protocol, but not under Thomas write rule.
B) It is not allowed under Basic time stamp order protocol, but allowed under Thomas write rule.
C) It is allowed under Basic time stamp order protocol as well as Thomas write rule.
D) It is not allowed under both Basic time stamp order protocol and Thomas write rule.

Answer

Option B

Question

14. If a transaction has obtained a _____ lock, it can read but cannot write on the item
a) Shared mode
b) Exclusive mode
c) Read only mode
d) Write only mode

Answer

Option A

Question

15. If a transaction has obtained a _____ lock, it can both read and write on the item
a) Shared mode
b) Exclusive mode
c) Read only mode
d) Write only mode

Answer

Option B

Question

17. If a transaction may obtain locks but may not release any locks then it is in _____ phase
a) Growing phase
b) Shrinking phase
c) Deadlock phase
d) Starved phase

Answer

Option A

Question

18.If a transaction may release locks but may not obtain any locks, it is said to be in _____ phase
a) Growing phase
b) Shrinking phase
c) Deadlock phase
d) Starved phase

Answer

Option B

Question

19.Which of the following protocols ensures conflict serializability and safety from deadlocks?
a) Two-phase locking protocol
b) Time-stamp ordering protocol
c) Graph based protocol
d) None of the mentioned

Answer

Option B

Question

20.Which of the following cannot be used to implement a timestamp
a) System clock
b) Logical counter
c) External time counter
d) None of the mentioned

Answer

Option C

Question

21. The default timestamp ordering protocol generates schedules that are
a) Recoverable
b) Non-recoverable
c) Starving
d) None of the mentioned

Answer

Option B, as dirty read is possible.

Question

Which of the following concurrency control protocols ensure both conflict serializability and freedom from deadlock?

I. 2-phase locking

II. Time-stamp ordering

GATE 2010

(A) I only

(B)II only

(C)

Both I and II

(D)

Neither I nor II

Answer

Option B

Question

In a database system, unique time stamps are assigned to each transaction using Lamport's logical clock. Let TS(T1) and TS(T2) be the time stamps of transactions T1 and T2 respectively. Besides, T1 holds a lock on the resource R, and T2 has requested a conflicting lock on the same resource R. The following algorithm is used to prevent deadlocks in the database assuming that a killed transaction is restarted with the same timestamp.

if TS(T2) <TS(T1) then T1 is killed else T2 waits.

Assume any transactions that is not killed terminates eventually. Which of the following is TRUE about the database system that uses the above algorithm to prevent deadlocks?

(A) The database system is both deadlock-free and starvation- free.

(B) The database system is deadlock- free, but not starvation-free.

(C) The database system is starvation-free but not deadlock- free.

(D) The database system is neither deadlock- free nor starvation-free.

Answer

This is basic TO protocol. Hence, we know it is deadlock free. However, the ordering of the transactions is done based on the timestamps which are static values are can't change. Hence, the DBMS is also starvation free. Thus, **Option A** is the correct answer.

Question

Which level of locking provides the highest degree of concurrency in a relational data base?
(A) Page
(B) Table
(C) Row
(D) Page, table and row level locking allow the same degree of concurrency

---

Answer

Explanation:
•Page level locking locks whole page i.e all rows therefore highly restrictive
•Table locking is mainly used for concurrency control with DDL operations
•A row share table lock is the least restrictive, and has the highest degree of concurrency for a table.It indicates the transaction has locked rows in the table and intends to update them.
Therefore, Row level provides highest level of concurrency. Option (C) is correct.

Question

Consider the following two phase locking protocol. Suppose a transaction T accesses (for read or write operations), a certain set of objects {O1,...,Ok}. This is done in the following manner:
Step 1. T acquires exclusive locks to O1, . . . , Ok in increasing order of their addresses.
Step 2. The required operations are performed.
Step 3. All locks are released.
This protocol will
(A) guarantee serializability and deadlock-freedom
(B) guarantee neither serializability nor deadlock-freedom
(C) guarantee serializability but not deadlock-freedom
(D) guarantee deadlock-freedom but not serializability

GATE 2016

Answer

Explanation: The above scenario is Conservative 2PL(or Static 2PL). In Conservative 2PL protocol, a transaction has to lock all the items it access before the transaction begins execution. It is used to **avoid deadlocks**. Also, 2PL is conflict serializable, therefore it **guarantees serializability.**
Therefore option A

Question

Calculate the order of leaf(p_{leaf}) and non leaf(p) nodes of a B+ tree based on the information given below
Search key field = 12 bytes Record pointer = 10 bytes Block pointer = 8 bytes Block size = 1 KB

(A)p_{leaf} = 51 & p = 46
(B)p_{leaf}= 47 & p = 52
(C)p_{leaf}= 46 & p = 50
(D)p_{leaf} = 52 & p = 47

Answer

A left node in a B+ tree is as follows –

Thus, we can write –

$$p_{leaf}(Keys + RP) + BP \leq Block\ Size$$

$$p_{leaf}(12 + 10) + 8 \leq 1024$$

$$\boldsymbol{p_{leaf} = 46}$$

Therefore, **Option C** is the correct answer

Question

Consider a database implemented using B+ tree for file indexing and installed on a disk drive with block size of 4 KB. The size of search key is 12 bytes and the size of tree/disk pointer is 8 bytes. Assume that the database has one million records. Also assume that no node of the B+ tree and no records are present initially in main memory. Consider that each record fits into one disk block. The minimum number of disk accesses required to retrieve any record in the database is ____.

Answer

Here, we need to check for **intermediate nodes** and first we will get the order of the nodes. An intermediate node looks like –

Thus, we get –

$$p(BP) + (p - 1)(Keys) \leq Block\ size$$

$$8p + 12p - 12 \leq 4096$$

$$\boldsymbol{p = 205}$$

Therefore, each node will have **204 keys** and **205 children.** Thus, we get –

LEVEL	NO OF NODES	NO OF KEYS	NO OF CHILDREN
1	1	204	205
2	205	41820	42025
3	42025	8573100	8615125

Therefore, we have a **3 – level B+ tree.** So we need to access 3 levels and then access the data from the node. Therefore, we need **4 memory access.**

Question

The following key values are inserted into a B^+-tree in which order of the internal nodes is 3, and that of the leaf nodes is 2, in the sequence given below. The order of internal nodes is the maximum number of tree pointers in each node, and the order of leaf nodes is the maximum number of data items that can be stored in it. The B^+- tree is initially empty.

10, 3, 6, 8, 4, 2, 1
The maximum number of times leaf nodes would get split up as a result of these insertions is

A 2

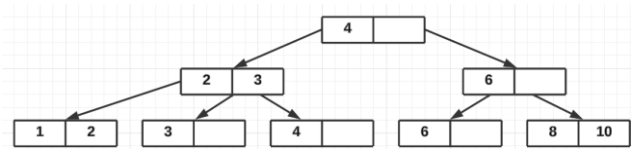
B 3

C 4

D 5

Answer

The insertion will be as follows –



There will be **4 splits** and thus **Option C** is the correct answer.

Question

The order of a leaf node in a B^+- tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?

A 63

B 64

C 67

D 68

Answer

Option A

Question

The order of an internal node in a B+ tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node?

A 24

B 25

C 26

D 27

Answer

Option C

Question

A B+ - tree index is to be built on the Name attribute of the relation STUDENT. Assume that all student names are of length 8 bytes, disk blocks are of size 512 bytes, and index pointers are of size 4 bytes. Given this scenario, what would be the best choice of the degree (i.e. the number of pointers per node) of the B+ - tree?

A 16

B 42

C 43

D 44

Answer

Option C

Question

Which one of the following statements is NOT correct about the B+ tree data structure used for creating an index of a relational database table?

A B+ tree is a height-balanced tree

B Non-leaf nodes have pointers to data records

C Each leaf node has a pointer to the next leaf node

D Key values in each node are kept in sorted order

Answer

Option B

Question

Consider a table T in a relational database with a key field K. A B-tree of order p is used as an access structure on K, where p denotes the maximum number of tree pointers in a B-tree index node. Assume that K is 10 bytes long; disk block size is 512 bytes; each data pointer P_D is 8 bytes long and each block pointer P_B is 5 bytes long. In order for each B-tree node to fit in a single disk block, the maximum value of p is

A 20

B 22

C 23

D 32

Answer

Option C

QUESTION BANK

Question 1

SQL is a non – procedural/declarative language in which we just need to provide the data to be queried but don't need to provide the method of querying. Is this true or false?

Question 2

Which of the following is the process of selecting the data storage and data access characteristics of the database?
(A)Logical DB design
(B)Physical DB design
(C)Testing & Performance tuning
(D)Schema Refinement

Question 3

Which is not the feature of DBMS?
(A)Data Redundancy
(B)Independence
(C)Flexibility
(D)Data Integrity

Question 4

A relational database contains two tables student and department in which student table has columns roll_no, name and dept_id and department table has columns dept_id and dept_name. The following insert statements were executed successfully to populate the empty tables:

```
Insert into department values (1, 'Mathematics')
Insert into department values (2, 'Physics')
Insert into student values (1, 'Navin', 1)
Insert into student values (2, 'Mukesh', 2)
Insert into student values (3, 'Gita', 1)
```

How many rows and columns will be retrieved by the following SQL statement?

```
Select * from student, department
```

(A) 0 row and 4 columns
(B) 3 rows and 4 columns
(C) 3 rows and 5 columns
(D) 6 rows and 5 columns

Question 5

Consider the following schedule:

```
S:R1(A), W2(A), Commit2, W1(A), W3(A), Commit3, Commit1
```

Which of the following is true?

(A) The schedule is view serializable schedule and strict recoverable schedule

(B) The schedule is non-serializable schedule and strict recoverable schedule

(C) The schedule is non-serializable schedule and is not strict recoverable schedule.

(D) The Schedule is serializable schedule and is not strict recoverable schedule

Question 6

In a database system, unique timestamps are assigned to each transaction using Lamport's logical clock. Let $TS(T_1)$ and $TS(T_2)$ be the timestamps of transactions T_1 and T_2 respectively. Besides, T_1 holds a lock on the resource R, and T_2 has requested a conflicting lock on the same resource R. The following algorithm is used to prevent deadlocks in the database system assuming that a killed transaction is restarted with the same timestamp.

if $TS(T_2) < TS(T_1)$ then

T_1 is killed

else T_2 waits.

Assume any transaction that is not killed terminates eventually. Which of the following is TRUE about the database system that uses the above algorithm to prevent deadlocks?

A. The database system is both deadlock-free and starvation-free.
B. The database system is deadlock-free, but not starvation-free.
C. The database system is starvation-free, but not deadlock-free.
D. The database system is neither deadlock-free nor starvation-free.

Question 7

Consider a file of 16384 records. Each record is 32 bytes long and its key field is of size 6 bytes. The file is ordered on a non-key field, and the file organization is unspanned. The file is stored in a file system with block size 1024 bytes, and the size of a block pointer is 10 bytes. If the secondary index is built on the key field of the file, and a multi-level index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multi-level index are respectively.

(A) 8 and 0

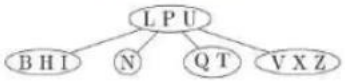
(B) 128 and 6

(C) 256 and 4

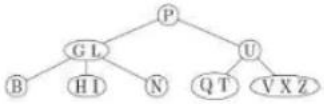
(D) 512 and 5

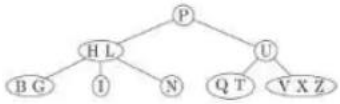
Question 8

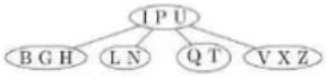
Consider the following $2-3-4$ tree (i.e., B-tree with a minimum degree of two) in which each data item is a letter. The usual alphabetical ordering of letters is used in constructing the tree.



What is the result of inserting G in the above tree?

A.


B.


C.


D. None of the above

Question 9

A B-tree of order 4 is built from scratch by 10 successive insertions. What is the maximum number of node splitting operations that may take place?

(A) 3

(B) 4

(C) 5

(D) 6

Question 10

A B-tree of order 4 is built from scratch by successive insertions of following keys in the given order.

10, 5, 14, 10, 3, 6, 30, 27, 9

What is the number of root node splitting operations that may take place with right biasing?

Question 11

A B-Tree used as an index for a large database table has four levels including the root node. If a new key is inserted in this index, then the maximum number of nodes that could be newly created in the process are?

ANSWER KEY

1	True								
2	B								
3	A								
4	D								

5	D								
6	A								
7	C								
8	B								
9	C								
10	1								
11	5								