DBMS 2019-20 Solution

Section A

Q.1 (a) What is Relational Algebra

Ans. Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- **Select:** It selects tuples that satisfy the given predicate from a relation.
- **Project:** It projects column(s) that satisfy a given predicate.
- **Union:** It performs binary union between two given relations
- **Set different:** The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
- **Cartesian product:** Combines information of two different relations into one.
- **Rename:** The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation

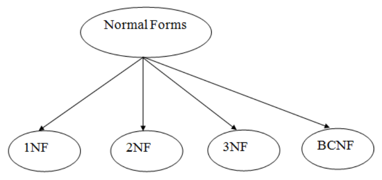
Q.(b) Explain Normalization. What is normal form?

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms:

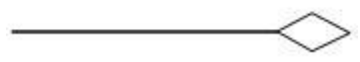


Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation 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.

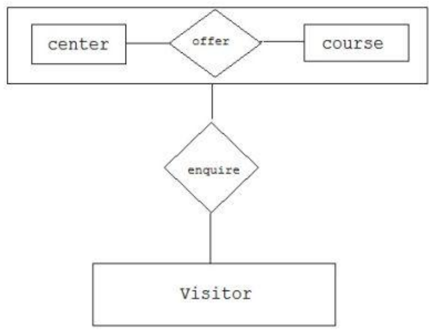
Q.(c) What do you mean by Aggregation?

- Aggregation refers to performing an operation on a group of values to get a single result.
- One limitation of the E-R model is that it cannot express relationships among the relationships so to overcome this we use aggregation.
- Aggregation is a process when the relation between two entity is treated as a single entity.
- Aggregation represents abstract entities by allowing relationship between relationships.
- Aggregation is a special type of Association.
- Aggregation is also called a "Has-a" relationship.

UML Notation:



Example



Q.(d) Define Super key, Candidate key, Primary Key & Foreign key.

Ans. Key plays an important role in relational database; it is used for identifying unique rows from table. It also establishes relationship among tables.

<u>Primary Key</u> – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

<u>Super Key</u> – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.

<u>Candidate Key</u> – A super key with no redundant attribute is known as candidate key

<u>Alternate Key</u> – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

<u>Composite Key</u> – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

<u>Foreign Key</u> – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Q.(e) What is Weak and Strong entity Set?

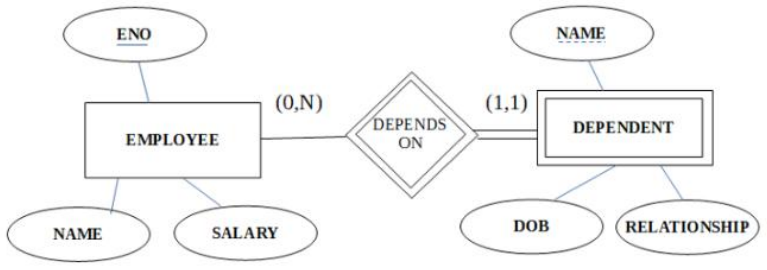
Ans. Weak Entity

A weak entity cannot be used independently as it is dependent on a strong entity type known as its owner entity. Also, the relationship that connects the weak entity to its owner identity is called the identifying relationship.

A weak entity always has a total participation constraint with respect to its identifying relationship because it cannot be identified independently of its owner identity.

A weak entity may have a partial key, which is a list of attributes that identify weak entities related to the same owner entity.

In the ER diagram, both the weak entity and its corresponding relationship are represented using a double line and the partial key is underlined with a dotted line.



In the given ER diagram, Dependent is the weak entity and it depends on the strong entity Employee via the relationship Depends on.

There can be an employee without a dependent in the Company but there will be no record of the Dependent in the company systems unless the dependent is associated with an Employee.

Strong Entity

A strong entity is complete by itself and is not dependent on any other entity type. It possess a primary key which describes each instance in the strong entity set uniquely. That means any element in the strong entity set can be uniquely identified.

A Strong entity is represented by a square with a single line unlike a Weak Entity which contained double lines.

Q.(f) What do you mean by conflict Serializable Schedule?

Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Note: Conflict pairs for the same data item are:
Read-Write
Write-Write
Write-Read

Q.(g) Define Concurrency Control.

Ans. Concurrency Control

- In the concurrency control, the multiple transactions can be executed simultaneously.
- It may affect the transaction result. It is highly important to maintain the order of execution of those transactions.

Problems of concurrency control

Several problems can occur when concurrent transactions are executed in an uncontrolled manner. Following are the three problems in concurrency control.

1. Lost updates
2. Dirty read
3. Unrepeatable read

1. Lost update problem

- When two transactions that access the same database items contain their operations in a way that makes the value of some database item incorrect, then the lost update problem occurs.
- If two transactions T1 and T2 read a record and then update it, then the effect of updating of the first record will be overwritten by the second update.

2. Dirty Read

- The dirty read occurs in the case when one transaction updates an item of the database, and then the transaction fails for some reason. The updated database item is accessed by another transaction before it is changed back to the original value.
- A transaction T1 updates a record which is read by T2. If T1 aborts then T2 now has values which have never formed part of the stable database.

3. Inconsistent Retrievals Problem

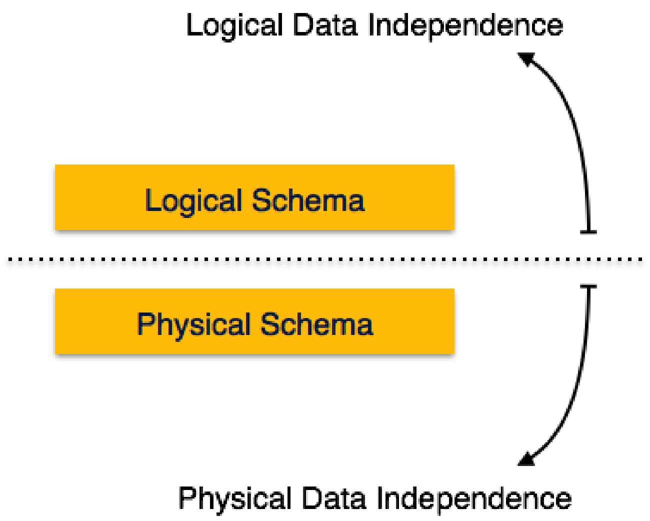
- Inconsistent Retrievals Problem is also known as unrepeatable read. When a transaction calculates some summary function over a set of data while the other transactions are updating the data, then the Inconsistent Retrievals Problem occurs.
- A transaction T1 reads a record and then does some other processing during which the transaction T2 updates the record. Now when the transaction T1 reads the record, then the new value will be inconsistent with the previous value.

SECTION B

Q.2 (a) Explain data independence with its types.

Ans. Data Independence

A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database. But as a DBMS expands, it needs to change over time to satisfy the requirements of the users. If the entire data is dependent, it would become a tedious and highly complex job.



Metadata itself follows a layered architecture, so that when we change data at one layer, it does not affect the data at another level. This data is independent but mapped to each other.

Logical Data Independence

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

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

Physical Data Independence

All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data.

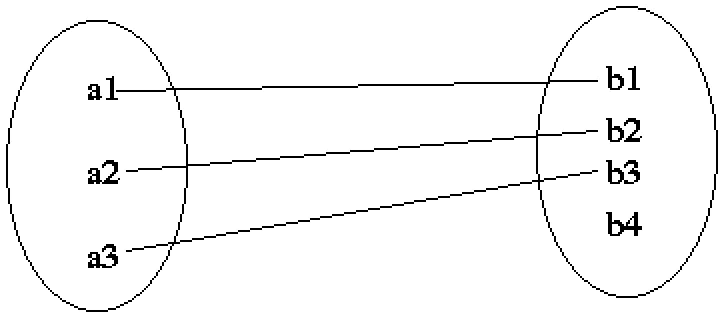
For example, in case we want to change or upgrade the storage system itself − suppose we want to replace hard-disks with SSD − it should not have any impact on the logical data or schemas.

Q.(b) Describe Mapping Constraints with its types.

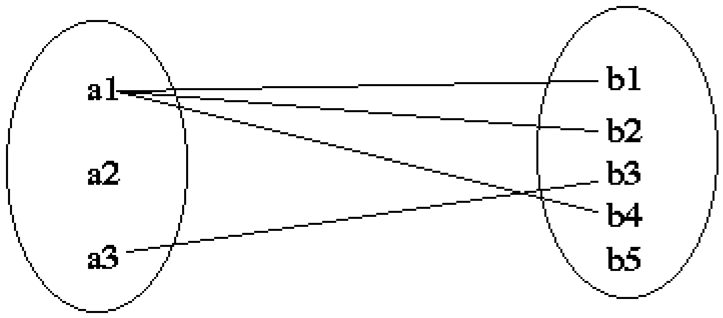
Ans. Mapping Constraints

- A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.
- It is most useful in describing the relationship sets that involve more than two entity sets.

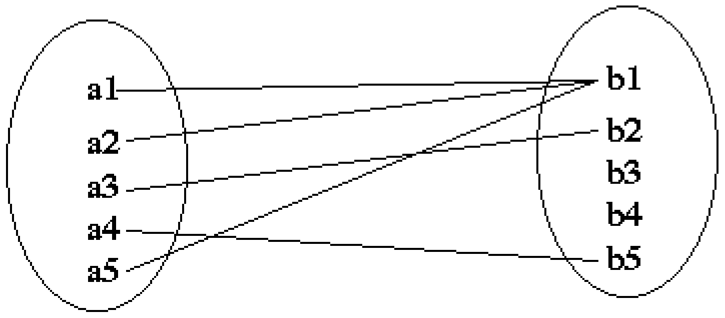
- For binary relationship set R on an entity set A and B, there are four possible mapping cardinalities. These are as follows:
  1. One to one (1:1)
  2. One to many (1:M)
  3. Many to one (M:1)
  4. Many to many (M:M)
- **one-to-one** - an entity in A is related to at most one entity in B, and an entity in B is related to at most one entity in A.



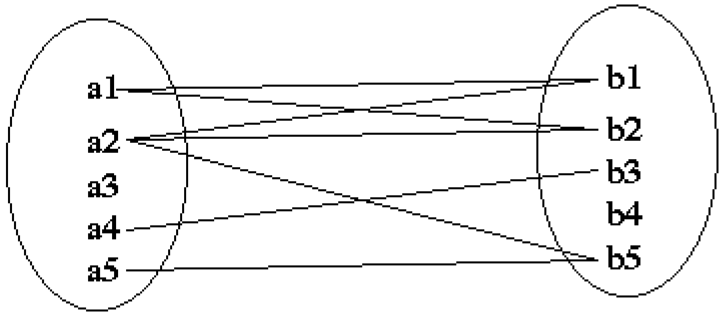
- **one-to-many** - an entity in A is related to any number of entities in B, but an entity in B is related to at most one entity in A.



- **many-to-one** - an entity in A is related to at most one entity in B, but an entity in B is related to any number of entities in A.



- **many-to-many** - an entity in A is related to any number of entities in B, but an entity in B is related to any number of entities in A.



Q.(c) Define Key. Explain various types of keys

Key plays an important role in relational database; it is used for identifying unique rows from table. It also establishes relationship among tables.

Types of keys in DBMS

<u>Primary Key</u> – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.
<u>Super Key</u> – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.
<u>Candidate Key</u> – A super key with no redundant attribute is known as candidate key
<u>Alternate Key</u> – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
<u>Composite Key</u> – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

Foreign Key – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables

Primary key in DBMS

Definition: A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

Example:

Student Table

Stu_Id	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Carl	22

In the above Student table, the Stu_Id column uniquely identifies each row of the table.

Note:

- We denote the primary key by underlining the column name.
- The value of primary key should be unique for each row of the table. Primary key column cannot contain duplicate values.
- Primary key column should not contain nulls.

Candidate Key in DBMS

A **super key** with no redundant attribute is known as candidate key. Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attributes. That's the reason they are also termed as minimal super key.

For example:

Emp_Id	Emp_Number	Emp_Name
E01	2264	Steve
E22	2278	Ajeet
E23	2288	Chaitanya
E45	2290	Robert

There are two candidate keys in above table:

{Emp_Id}

{Emp_Number}

Note: A **primary key** is being selected from the group of candidate keys. That means we can either have Emp_Id or Emp_Number as primary key.

Alternate key in DBMS

Out of all **candidate keys**, only one gets selected as **primary key**, remaining keys are known as alternative or secondary keys.

For example: Consider the below table

Emp_Id	Emp_Number	Emp_Name
E01	2264	Steve
E22	2278	Ajeet
E23	2288	Chaitanya
E45	2290	Robert

There are two candidate keys in above table:
{Emp_Id}
{Emp_Number}

Since we have selected Emp_Id as primary key, the remaining key Emp_Number would be called alternative or secondary key.

Super key in DBMS

Definition: A super key is a set or one of more columns (attributes) to uniquely identify rows in a table. Often people get confused between super key and candidate key, so we will also discuss a little about candidate key here.

How candidate key is different from super key?
Answer is simple – Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attribute. That's the reason they are also termed as minimal super key.

Let's take an example to understand this: **Employee table**

Emp_SSN	Emp_Number	Emp_Name
123456789	226	Steve
999999321	227	Ajeet
888997212	228	Chaitanya
777778888	229	Robert

Super keys:

- {Emp_SSN}
- {Emp_Number}
- {Emp_SSN, Emp_Number}
- {Emp_SSN, Emp_Name}
- {Emp_SSN, Emp_Number, Emp_Name}
- {Emp_Number, Emp_Name}

All of the above sets are able to uniquely identify rows of the employee table.

Foreign key in DBMS

Definition: Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

For example:
In the below example the Stu_Id column in Course_enrollment table is a foreign key as it points to the primary key of the Student table.

Course_enrollment table:

Course_Id	Stu_Id
C01	101
C02	102
C03	101
C05	102
C06	103
C07	102

Student table:

Stu_Id	Stu_Name	Stu_Age
101	Chaitanya	22
102	Arya	26
103	Bran	25
104	Jon	21

Q.(d) Explain the Phantom Phenomena. Discuss a time stamp Protocol that avoids the phantom phenomena.

Ans. Phantom problem is a phenomena. A data problem during concurrency update.

In the phantom problem, a transaction accesses a relation more than once with the same predicate in the same transaction, but sees new phantom tuples on re-access that were not seen on the first access.

In other words, a transaction reruns a query returning a set of rows that satisfies a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition

Timestamp-Based Protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction T_i is denoted as $TS(T_i)$.
- Read time-stamp of data-item X is denoted by R-timestamp(X).
- Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows −

- **If a transaction Ti issues a read(X) operation −**
    - If TS(Ti) < W-timestamp(X)
        - Operation rejected.
    - If TS(Ti) >= W-timestamp(X)
        - Operation executed.
    - All data-item timestamps updated.
- **If a transaction Ti issues a write(X) operation −**
    - If TS(Ti) < R-timestamp(X)
        - Operation rejected.
    - If TS(Ti) < W-timestamp(X)
        - Operation rejected and Ti rolled back.
    - Otherwise, operation executed.

Thomas' Write Rule

This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and T_i is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making T_i rolled back, the 'write' operation itself is ignored.

Q.(e) What are distributed Database? List advantage and Disadvantage of data replication and fragmentation

Ans. Distributed Database System

A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e, on multiple computers or over a network of computers. A distributed database system is located on various sited that don’t share physical components. This maybe required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.

Types:
1. Homogeneous Database:
In a homogeneous database, all different sites store database identically. The operating system, database management system and the data structures used – all are same at all sites. Hence, they're easy to manage.

2. Heterogeneous Database:
In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

Data Replication

Data replication is the process of storing separate copies of the database at two or more sites. It is a popular fault tolerance technique of distributed databases.

Advantages of Data Replication

- **Reliability** − In case of failure of any site, the database system continues to work since a copy is available at another site(s).
- **Reduction in Network Load** − Since local copies of data are available, query processing can be done with reduced network usage, particularly during prime hours. Data updating can be done at non-prime hours.
- **Quicker Response** − Availability of local copies of data ensures quick query processing and consequently quick response time.
- **Simpler Transactions** − Transactions require less number of joins of tables located at different sites and minimal coordination across the network. Thus, they become simpler in nature.

Disadvantages of Data Replication

- **Increased Storage Requirements** − Maintaining multiple copies of data is associated with increased storage costs. The storage space required is in multiples of the storage required for a centralized system.
- **Increased Cost and Complexity of Data Updating** − Each time a data item is updated, the update needs to be reflected in all the copies of the data at the different sites. This requires complex synchronization techniques and protocols.
- **Undesirable Application – Database coupling** − If complex update mechanisms are not used, removing data inconsistency requires complex co-ordination at application level. This results in undesirable application – database coupling.

Fragmentation

Fragmentation is the task of dividing a table into a set of smaller tables. The subsets of the table are called **fragments**. Fragmentation can be of three types: horizontal, vertical, and hybrid (combination of horizontal and vertical). Horizontal fragmentation can further be classified into two techniques: primary horizontal fragmentation and derived horizontal fragmentation.

Fragmentation should be done in a way so that the original table can be reconstructed from the fragments. This is needed so that the original table can be reconstructed from the fragments whenever required. This requirement is called “reconstructiveness.”

Advantages of Fragmentation

- Since data is stored close to the site of usage, efficiency of the database system is increased.
- Local query optimization techniques are sufficient for most queries since data is locally available.
- Since irrelevant data is not available at the sites, security and privacy of the database system can be maintained.

Disadvantages of Fragmentation

- When data from different fragments are required, the access speeds may be very high.
- In case of recursive fragmentations, the job of reconstruction will need expensive techniques.
- Lack of back-up copies of data in different sites may render the database ineffective in case of failure of a site.

SECTION C

Q.3 (a) Define Join. Explain different types of join.

Ans. DBMS Joins

We understand the benefits of taking a Cartesian product of two relations, which gives us all the possible tuples that are paired together. But it might not be feasible for us in certain cases to take a Cartesian product where we encounter huge relations with thousands of tuples having a considerable large number of attributes.

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

We will briefly describe various join types in the following sections.

Theta (θ) Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.

Notation

$R1 \bowtie_{\theta} R2$

R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don't have anything in common, that is $R1 \cap R2 = \Phi$.

Theta join can use all kinds of comparison operators.

Student

SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects

Class	**Subject**
10	Math
10	English
11	Music
11	Sports

Student_Detail −

$\text{STUDENT} \bowtie_{\text{Student.Std = Subject.Class}} \text{SUBJECT}$

Student_detail

SID	**Name**	**Std**	**Class**	**Subject**
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses

CID	Course	Dept
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HoD

Dept	Head
CS	Alex
ME	Maya
EE	Mira

Courses ⋈ HoD

Dept	CID	Course	Head
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Outer Joins

Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins − left outer join, right outer join, and full outer join.

Left Outer Join(R ⟕S)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Left

A	B
100	Database
101	Mechanics

102	Electronics

Right

A	B
100	Alex
102	Maya
104	Mira

Courses ⟕HoD

A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

Right Outer Join: (R ⟖S)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Courses ⟖HoD

A	B	C	D
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: (R ⟗S)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses ⟗HoD

A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

Q.(b) Disscuss the following terms (i) DDL Command (ii) DML Command

DDL

Data Definition Language (DDL) statements are used to define the database structure or schema. Some examples:

- CREATE - to create objects in the database
- ALTER - alters the structure of the 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

DML

Data Manipulation Language (DML) statements are used for managing data within schema objects. Some examples:

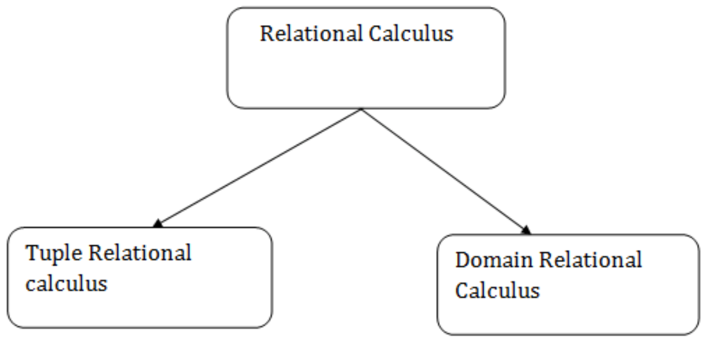
- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency

Q.4 (a) What is tuple relational calculus and domain relational calculus.

Ans. **What is Relational Calculus?**

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn't tell how to retrieve it.

Types of Relational Calculus



1. Tuple Relational Calculus (TRC)

Tuple relational calculus is used for selecting those tuples that satisfy the given condition.
Table: Student

```
First_Name     Last_Name     Age
----------     ---------     ----
Ajeet            Singh           30
Chaitanya     Singh           31
Rajeev          Bhatia           27
Carl              Pratap           28
```

Lets write relational calculus queries.

Query to display the last name of those students where age is greater than 30

```
{ t.Last_Name | Student(t) AND t.age > 30 }
```

In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part we specify the fields which we want to display for the selected tuples.

The result of the above query would be:

```
Last_Name
---------
Singh
```

Query to display all the details of students where Last name is ‘Singh’

```
{ t | Student(t) AND t.Last_Name = 'Singh' }
```

Output:

```
First_Name     Last_Name     Age
----------     ---------     ----
Ajeet                  Singh        30
Chaitanya           Singh        31
```

2. Domain Relational Calculus (DRC)

In domain relational calculus the records are filtered based on the domains.
Again we take the same table to understand how DRC works.
Table: Student

```
First_Name     Last_Name     Age
----------     ---------     ----
Ajeet                  Singh        30
Chaitanya           Singh        31
Rajeev                Bhatia       27
```

```
Carl                    Pratap        28
```

Query to find the first name and age of students where student age is greater than 27

```
{< First_Name, Age > | ∈ Student ∧ Age > 27}
```

Note:
The symbols used for logical operators are: ∧ for AND, ∨ for OR and ┓ for NOT.

Output:

```
First_Name     Age
----------     ----
Ajeet            30
Chaitanya        31
Carl             28
```

Difference between Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC) :

TUPLE RELATIONAL CALCULUS (TRC)	DOMAIN RELATIONAL CALCULUS (DRC)
In TRS, the variables represent the tuples from specified relation.	In DRS, the variables represent the value drawn from specified domain.
A tuple is a single element of relation.In database term, it is a row.	A domain is equivalent to column data type and any constraints on value of data.
In this filtering variable uses tuple of relation.	In this filtering is done based on the domain of attributes.
Notation : {T | P (T)} or {T | Condition (T)}	Notation : { a1, a2, a3, …, an | P (a1, a2, a3, …, an)}
Example : {T | EMPLOYEE (T) AND T.DEPT_ID = 10}	Example : { | < EMPLOYEE > DEPT_ID = 10 }

Q.(b) Describe the following terms (i) Multivalued dependency (ii) Trigger

Ans (i) Multivalued dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL $\rightarrow\rightarrow$ MANUF_YEAR
2. BIKE_MODEL $\rightarrow\rightarrow$ COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

(ii) **Triggers**

Triggers are the SQL statements that are **automatically executed** when there is any change in the database. The triggers are executed **in response to certain events** (INSERT, UPDATE or DELETE) in a particular table. These triggers help in maintaining the integrity of the data by changing the data of the database in a systematic fashion.

Syntax

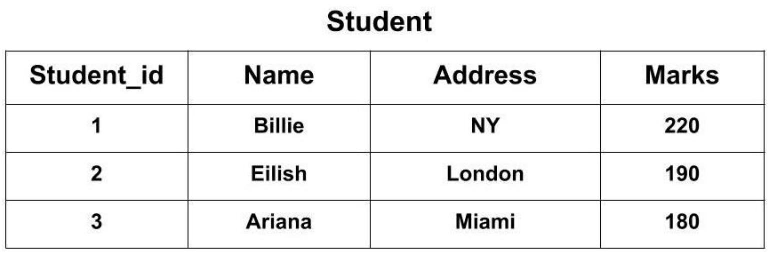
create trigger **Trigger_name**

```
(before | after)
[insert | update | delete]
on [table_name]
[for each row]
[trigger_body]
```

1. **CREATE TRIGGER:** These two keywords specify that a triggered block is going to be declared.
2. **TRIGGER_NAME:** It creates or replaces an existing trigger with the Trigger_name. The trigger name should be unique.
3. **BEFORE | AFTER:** It specifies when the trigger will be initiated i.e. before the ongoing event or after the ongoing event.
4. **INSERT | UPDATE | DELETE**: These are the DML operations and we can use either of them in a given trigger.
5. **ON[TABLE_NAME]:** It specifies the name of the table on which the trigger is going to be applied.
6. **FOR EACH ROW:** Row-level trigger gets executed when any row value of any column changes.
7. **TRIGGER BODY:** It consists of queries that need to be executed when the trigger is called.

Example

Suppose we have a table named **Student** containing the attributes *Student_id, Name, Address, and Marks.*



Student

Student_id	Name	Address	Marks
1	Billie	NY	220
2	Eilish	London	190
3	Ariana	Miami	180

Now, we want to create a **trigger** that will add 100 marks to each new row of the *Marks* column whenever a new student is inserted to the table.

The SQL Trigger will be:

```
CREATE TRIGGER Add_marks
BEFORE
INSERT
```

```
ON Student
FOR EACH ROW
SET new.Marks = new.Marks + 100;
```

The new keyword refers to the row that is getting affected.

After creating the trigger, we will write the **query for inserting a new student** in the database.

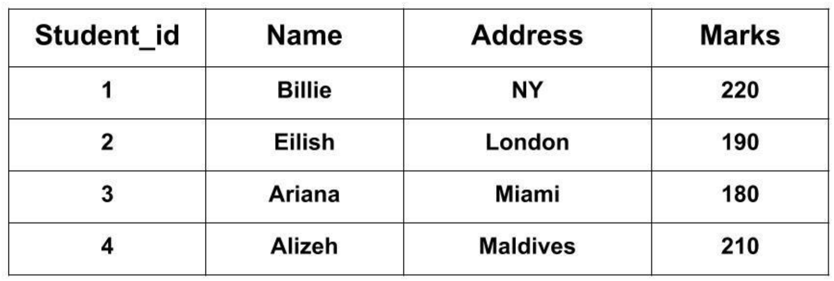
```
INSERT INTO Student(Name, Address, Marks) VALUES('Alizeh', 'Maldives', 110);
```

*The **Student_id column** is an auto-increment field and will be generated automatically when a new record is inserted into the table.*

To see the final output the query would be:

```
SELECT * FROM Student;
```

Student



Student_id	Name	Address	Marks
1	Billie	NY	220
2	Eilish	London	190
3	Ariana	Miami	180
4	Alizeh	Maldives	210

Q.5 (a) What do you understand by ACID Properties of Transaction? Explain in detail.

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A’s Account

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

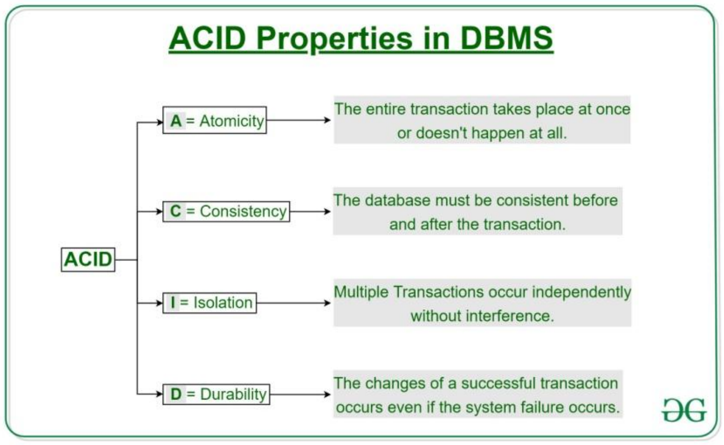
B’s Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.


ACID Properties in DBMS
ACID
A = Atomicity
The entire transaction takes place at once or doesn't happen at all.
C = Consistency
The database must be consistent before and after the transaction.
I = Isolation
Multiple Transactions occur independently without interference.
D = Durability
The changes of a successful transaction occurs even if the system failure occurs.

Q5 (b) Discuss about Deadlock Prevention schemes.

Deadlock Prevention

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Wait-Die Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur −

- If $TS(T_i) < TS(T_j)$ − that is T_i, which is requesting a conflicting lock, is older than T_j − then T_i is allowed to wait until the data-item is available.
- If $TS(T_i) > TS(t_j)$ − that is T_i is younger than T_j − then T_i dies. T_i is restarted later with a random delay but with the same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

Wound-Wait Scheme:

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.
- If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur −

- If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back − that is T_i wounds T_j. T_j is restarted later with a random delay but with the same timestamp.
- If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available.

This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.

In both the cases, the transaction that enters the system at a later stage is aborted.

Another group of protocols which prevents deadlock but *does not require Timestamps*. They are discussed below:

- **No-waiting Algorithm :** This follows a simple approach, if a Transaction is unable to obtain a lock, it is immediately aborted and then restarted after a certain time delay without

checking if a deadlock will occur or not. Here, no Transaction ever waits so there is no possibility for deadlock.
This method is somewhat not practical. It may cause transaction to abort and restart unnecessarily.

- **Cautious Waiting :** If $\mathbf{T_i}$ tries to lock an item *X* but is not able to do because *X* is locked by some $\mathbf{T_j}$. In such a conflict, if $\mathbf{T_j}$ is not waiting for some other locked item, then $\mathbf{T_i}$ is allowed to wait, otherwise *abort* $\boldsymbol{T_i}$.

Q6 (a): What is concurrency control? Why it is needed in database systems?

Sol. Concurrency control is the procedure in DBMS for managing simultaneous operations without conflicting with each another. Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical database, would have a mix of reading and WRITE operations and hence the concurrency is a challenge.

Concurrency control is used to address such conflicts which mostly occur with a multi-user system. It helps you to make sure that database transactions are performed concurrently without violating the data integrity of respective databases.

Here, are some issues which you will likely to face while using the Concurrency Control method:

- **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
- Uncommitted dependency issues occur when the second transaction selects a row which is updated by another transaction (**dirty read**)
- **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.
- **Incorrect Summary issue** occurs when one transaction takes summary over the value of all the instances of a repeated data-item, and second transaction update few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

Reasons for using Concurrency control method is DBMS:

- To apply Isolation through mutual exclusion between conflicting transactions
- To resolve read-write and write-write conflict issues
- To preserve database consistency through constantly preserving execution obstructions
- The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
- Concurrency control helps to ensure serializability

Q 6 (b) Give the following queries in relational algebra using the relational schema:

Student(id, name)

Enrolled(id, code)

Subject(code, lecturer)

i) What are the names of students enrolled in cs3020?

Sol: $\pi_{name}(\sigma_{cs3020=code}(student \bowtie enrolledIn))$

ii) Which subjects is Hector taking?

Sol: $\pi_{code}(\sigma_{name=Hector}(student \bowtie enrolledIn))$

iii) Who teaches cs1500?

Sol: $\pi_{lecturer}(\sigma_{code=cs1500}(subject))$

iv) Who teaches cs1500 or cs3020?

Sol: $\pi_{lecturer}(\sigma_{code=cs1500 \vee code=cs3020}(subject))$

v) Who teaches atleast two different subjects?

Sol: $\pi_{lecturer}(\sigma_{R.lecturer = S.lecturer \text{ AND } R.code = S.code}(R \bowtie S))$

Note: For this query we have to relate subject to itself. To disambiguate the relation, we will call the subject relation R or S.

vi) What are the names of students in cs1500 or cs307?

Sol: $\pi_{name}(\sigma_{code=cs1500}(student \bowtie enrolledIn)) \cup \pi_{name}(\sigma_{code=cs307}(student \bowtie enrolledIn))$

vii) What are the names of students in both cs1500 and cs1200?

Sol: $\pi_{name}(\sigma_{code=cs1500}(student \bowtie enrolledIn)) \cap$

$\pi_{name}(\sigma_{code=cs1200}(student \bowtie enrolledIn))$

Q7(a). Explain Directory System in detail.

Sol. A directory is a listing of information about some class of objects such as persons. Directories can be used to find information about a specific object, or in the reverse direction to find objects that meet a certain requirement.

Directory Access Protocols

Directory information can be made available through Web interfaces, as many organizations, and phone companies in particular do. Such interfaces are good for humans. However, programs too, need to access directory information. Directories can be used for storing other types of information, much like file system directories. Lightweight Directory Access Protocol (LDAP) have been developed to provide a standardized way of accessing data in a directory.

LDAP: Lightweight Directory Access Protocol

In general a directory system is implemented as one or more servers, which service multiple clients. Clients use the application programmer interface defined by directory system to communicate with the directory servers. Directory access protocols also define a data model and access control.
The X.500 directory access protocol, defined by the International Organization for Standardization (ISO), is a standard for accessing directory information. However, the protocol is rather complex, and is not widely used. The Lightweight Directory Access Protocol (LDAP) provides many of the X.500 features, but with less complexity, and is widely used.

LDAP Data Model

In LDAP directories store entries, which are similar to objects. Each entry must have a distinguished name (DN), which uniquely identifies the entry. A DN is in turn made up of a sequence of relative distinguished names (RDNs). For example, an entry may have the following distinguished name.

cn=Silberschatz, ou=Bell Labs, o=Lucent, c=USA

LDAP allows the definition of object classes with attribute names and types. Inheritance can be used in defining object classes. Moreover, entries can be specified to be of one or more object classes. It is not necessary that there be a single most-specific object class to which an entry belongs.
Entries are organized into a directory information tree (DIT), according to their distinguished names. Entries at the leaf level of the tree usually represent specific objects. Entries that are internal nodes represent objects such as organizational units, organizations, or countries. The children of a node have a DN containing all the RDNs of the parent, and one or more additional RDNs.

Data Manipulation

LDAP defines a network protocol for carrying out data definition and manipulation. Users of LDAP can either use an application programming interface, or use tools provided by various vendors to perform data definition and manipulation. LDAP also defines a file format called LDAP Data Interchange Format (LDIF) that can be used for storing and exchanging information.

Q7 (b) Consider the following relational database. Give an expression in SQL for each of the following queries uniderline records are primary key.

Employee(person_name, street, city)

Works(person_name, company_name, salary)

Company(company_name, city)

Manages(person_name, Manager_name)

i) Find the name of all employees who works for ABC bank.

Sol: Select person_name from Works Where company_name='ABC';

ii) Find the name of all employees who live in the same city and on the same street as do their managers.

Sol: Select E1.person_name From Employee as E1, Employee as E2, Manages as M Where E1.person_name=M.person_name and E2.person_name=M.manager_name and E1.stree=E2.street and E1.city=E2.city;

iii) Find the name street address and cities of residence of all employees who work for ABC bank and earn more than 7000 per annum.

Sol: select E.person_name, street, city from Employee as E, Works as W where E.person_name=W.person_name and W.company_name=’ABC’ and W.salary>7000;

iv) Find the name of all employees who earn more than every employee of XYZ.

Sol: select person_name from Works where salary>(select max(salary) from Works where company_name=’XYZ’);

v) Give all employees of corporation ABC a 7% salary rise.

Sol: update Works set salary=salary*1.07 where company_name=’ABC’;

vi) Delete all tuples in the works relation for employees of ABC.

Sol: delete from Works where company_name=’ABC’;

vii) Find the name of all employees in this DATABASE who live in the same city as the company for which they work.

Sol: select E.person_name from Employee as E, Works as W, Company as C where E.person_name=W.person_name and E.city=C.city and W.company_name=C.company_name;