DATABASE MANAGEMENT SYSTEM

IV C.S.

1 INTRODUCTION TO DATABASE SYSTEMS

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 ***Why E-R model used in DBMS?*** *[R.T.U. 2019]*

Ans. The Entity-Relationship Model : The entity-relationship (E-R) data model was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database. The E-R data model is one of several semantic data models; the semantic aspect of the model lies in its representation of the meaning of the data.

Q.2 ***What do you mean by Referential Integrity?*** *[R.T.U. 2019]*

Ans. Referential Integrity : This constraints identifies any column in the same table or between different tables. For a column to be defined as a foreign key, it should be a defined as a primary key in table.

Which it is referring. One or more columns can be defined as foreign key.

Syntax to define a foreign key at column level.

[CONSTRAINT Constraint_name] REFERENCES

Referenced. Table_name (column_name)

Syntax to define a foreign key at table laud.

[CONSTRAINT constraint_name] FOREIGN KEY (Column_name) references referenced_table name (column_name)

Q.3 ***Explain the concepts of Primary key.*** *[R.T.U. 2019]*

Ans. Every record must have one unique identifier called the primary key that has a unique value within the table or collection. Primary keys can be concatenated which means that the uniqueness can be made up of one or more fields.

Syntax to define a primary key at column level :

Column name data type [CONSTRAINT constraint_name] PRIMARY KEY

Syntax to define a primary key at table level :

[CONSTRAINT Constraint_name] PRIMARY KEY

[Column_name1, column_name2]

Q.4 ***What is Entity? Explain with a suitable example.*** *[R.T.U. 2019]*

Ans. Entity : In database systems, objects are referred as entity. *"An entity is a thing or object in the real world that is distinguishable from other things or objects."*

For example, each person is an entity and banks accounts can be considered as entity.

Note : Object and class of OOPs are referred as an entity and an entity set in database system respectively.

Q.5 ***What is DBMS?*** *[R.T.U. 2019]*

Ans. DBMS : A Database Management system is a collection of interrelated data and collection of programs to access that data. The data describes one particular enterprise. Database systems are ubiquitous today and most people interact either directly or indirectly, with

databases many times everyday. A major purpose of the database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

Underlying the structure of a database is the data model- a collection of conceptual tools for describing data, data relationships, data semantics and data constraints. The entity relationship (E-R) data model is a widely used data model which provides a convenient graphical representation to view data, relationship and constraints.

Q.6 ***What is the difference between logical data independence and physical data independence?*** *[R.T.U. 2016, 2015]*

Ans.

S. No.	Logical Data Independence	Physical Data Independence
1.	Indicates that conceptual/logical schema can be changed without affecting the existing external (view) schemas.	Indicates that the physical storage structures or devices used for storing the data could be changed without any change in the conceptual view or external view.
2.	The change would be absorbed by the mapping between the external and conceptual (logical) levels.	The change would be absorbed by the mapping between the conceptual and internal levels.
3.	Modifications such as the deletion of a conceptual view field or record may require changes in the external view and application program.	Modifications of physical level improve the performance.

Q.7 ***What do you mean by constraints? Explain different types of constraints with examples.*** *[R.T.U. 2016]*

Ans. Constraints : Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the data integrity during an update/delete/insert into a table.

Different Types of constraints that can be created in RDBMS

1. Mapping Constraints :

Mapping Cardinalities : Express the number of entities to which another entity can be associated via a relationship. For binary relationship sets between entity sets A and B, the mapping cardinality must be one of:

(i) **One-to-one :** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

(ii) **One-to-many :** An entity in A is associated with any number in B. An entity in B is associated with at most one entity in A.

(iii) **Many-to-one :** An entity in A is associated with at most one entity in B. An entity in B is associated with any number in A.

(iv) **Many-to-many :** Entities in A and B are associated with any number from each other.

Q.8 ***Explain physical and external view of data.***

Ans. External View : In the relational model, the external view also presents data as a set of relations. It is tailored to the needs of a particular category of users. Portions of stored data should not be seen by some users and begins to implement a level of security and simplifies the view for these users. For example, students should not see faculty salaries, faculty should not see billing or payment data, etc.

Physical View : The physical view describes the details of how data is stored: files, indices, etc., on the random access disk system. It also typically describes the record layout of files and type of files (hash, b-tree, flat).

Q.9 ***Differentiate between candidate keys and super keys.***

Ans. Difference between Candidate Keys and Super Keys

S.No.	Candidate Keys	Super Keys
1.	A candidate key is any set of one or more columns whose combined value is unique through out the table.	An attribute, or group of attributes, that is sufficient to distinguish every tuple in the relation from every other one.
2.	Each candidate key is not called super key and no component of a candidate key is allowed to be null.	Each super key is called a candidate key.

Q.10 ***Explain the difference between partial & total participation.***

Ans. Difference between Partial and Total Participation

S.No.	Partial Participation	Total Participation
1.	All instances need not participate.	Every entity instance must be connected through the relationship to another instance of the other participating entity types.
2.	Represented by single line from entity rectangle to relationship diamond.	Represented by double line from entity rectangle to relationship diamond.

Q.11 ***Differentiate between entity and entity sets.***

Ans. Difference between Entity and Entity Sets

S. No.	Entity	Entity Set
1.	Entities are the people, places, things, events and concepts of interest to an organization.	Entity set represents collection of things. Example : An employee entity set might represent a collection of all the employees that work for an organization.
2.	Entity can be concrete like employee, book, project etc. and can be abstract or a concept. Example : title, job company etc.	Entity set need not be disjoint. For example : the entity set employee (all employee of bank) and the entity set customer (all customer of the bank) may have members in common.

PART-B

Q.12 ***What is E-R model? What are the features of E-R model? Draw and explain E-R model for Library Management System.*** *[R.T.U. 2019]*

Ans. E-R Model : The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities* and of *relationships* among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. For example, each person is an entity and bank accounts can be considered as entities.

Entities are described in a database by a set of **attributes**. For example, the attributes *account-number* and *balance* may describe one particular account in a bank and they form attributes of the *account* entity set. Similarly, attributes *customer-name*, *customer-street* address and *customer-city* may describe a *customer* entity.

An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address and city).

A unique customer identifier must be assigned to each customer. In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.

A **relationship** is an association among several entities. For example, a *depositor* relationship associates a customer with each account that she has. The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.

The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram*, which is built up from the following components -

(i) **Rectangles :** Which represent entity sets.
(ii) **Ellipses :** Which represent attributes.
(iii) **Diamonds :** Which represent relationships among entity sets.
(iv) **Lines :** Which link attributes to entity sets and entity sets to relationships.

Each component is labeled with the entity or relationship that it represents.

As an illustration, consider part of a database banking system consisting of customers and of the accounts that these customers have. Figure 1 shows the corresponding E-R diagram.

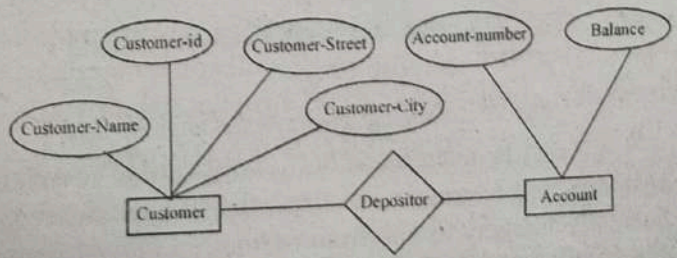


Fig. 1 : A simple E-R Diagram

The E-R diagram indicates that there are two entity sets, *customer* and *account*, with attributes. The diagram also shows a relationship *depositor* between customer and account.

In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform. One important constraint is **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set. For example - if each account must belong to only one customer, the E-R model can express that constraint.

E-R diagrams also provide a way to indicate more complex constraints on the number of times each entity participates in relationships in a relationship set. An edge between an entity set and a binary relationship set can have an associated minimum and maximum cardinality, shown in the form *l..h*, where *l* is the minimum and *h* is the maximum cardinality. A minimum value of 1 indicates total participation of the entity set in the relationship set. A maximum value of 1 indicates that the entity participates in at most one relationship, while a maximum value * indicates no limit.

Note that a label 1..* on an edge is equivalent to a double line.

For example, consider Figure 2. The edge between *loan* and *borrower* has a cardinality constraint of 1..1, meaning the minimum and the maximum cardinality both are 1. That is, each loan must have exactly one associated customer. The limit 0..* on the edge from *customer* to *borrower* indicates that a customer can have zero or more loans. Thus, the relationship *borrower* is one to many from *customer* to *loan* and further the participation of *loan* in *borrower* is total.

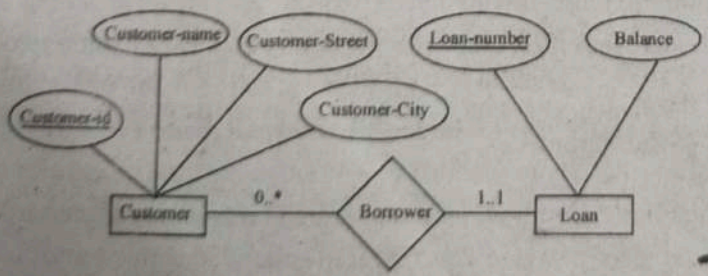


Fig. 2 : Cardinality Limits on Relationship Sets

It is easy to misinterpret the 0..* on the edge between *customer* and *borrower* and think that the relationship *borrower* is many to one from *customer* to *loan*-this is exactly the reverse of the correct interpretation.

If both edges from a binary relationship have a maximum value of 1, the relationship is one to one. If we had specified a cardinality limit of 1..* on the edge between *customer* and *borrower*, we would be saying that each customer must have at least one loan.

E-R Notations :

1. E entity set =

customer , employee , loan , branch , account

These are entity sets.

2. weak entity → Loan payment are weak entity.

3. Relationship set → Shows the relationship

4. A ← Show primary set

Customer_id, branch name, account-no. These are primary keys.

5. ( ) – All the attributes are shown through ellipse.

6. A – Derived attribute

Employment-length is derived attribute.

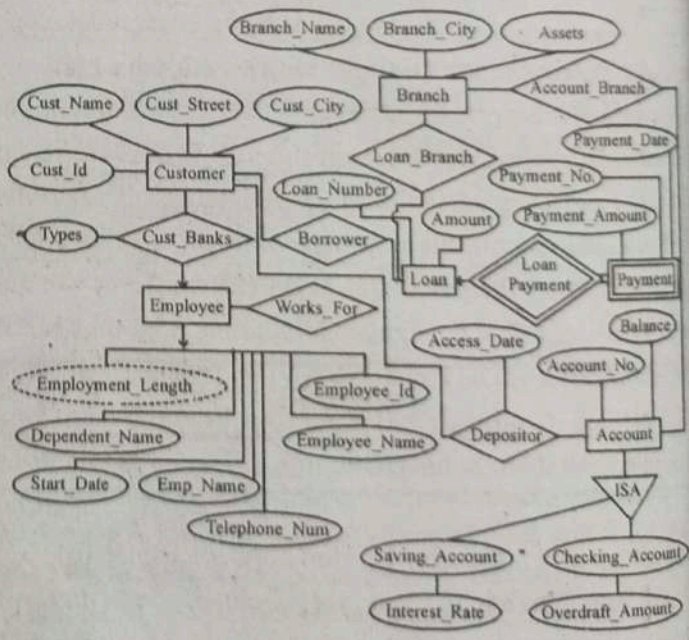
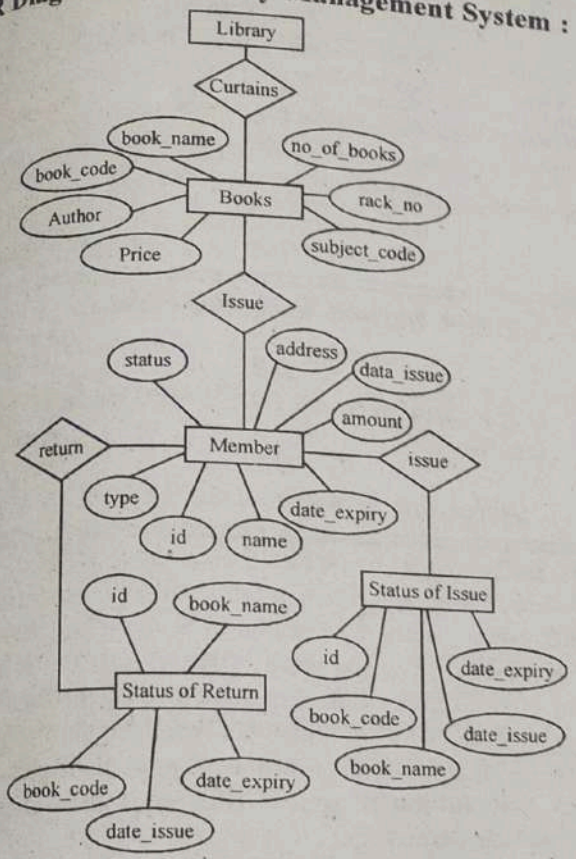


Fig. 3 : E-R Diagram

The E-R model defines the conceptual view of a database. E-R model is a high-level conceptual model for database design.

ER Diagram for Library Management System :



Useful in designing Real World Database

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

Then we have the notion of entity sets. An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

It is this important aspect of the entities and the design of E-R Model, that make the Entity-Relationship Model useful in designing Real-World Databases, because real-world objects can directly be taken as entities. There real-world properties such as age or name for a student can directly be taken as attributes for entities. Thus, E-R Model is useful for designing real world databases.

Convert ER-Diagram into Tables :

1. Members :

Column Name	Id_no (PK)	Name	Address	Date of Issue	Date of Expiry	Status
Data Type	Text	Text	Text	Date/time	Date/time	Text

2. Add_Books :

Column Name	Book_name	Book_code (PK)	Author	Date_of_arrival	Price	Rack_no	No_of_books	Subject_code
Data Type	Text	Text	Text	Date/time	Date/time	Text	Text	Text

3. Issue :

Column Name	Id_no (FK)	Book_Name	Issue_date	Due_date
Data Type	Text	Text	Date/time	Date/time

Q.13 ***Differentiate between file system and DBMS. Explain the ternary relationship with a suitable example.*** *[R.T.U. 2019]*

Ans. File System vs DBMS - Difference between File System and DBMS

File Management System	Database Management System
File System is a general, easy-to-use system to store general files which require less security and constraints.	Database management system is used when security constraints are high.
Data Redundancy is more in file management system.	Data Redundancy is less in database management system.
Data Inconsistency is more in file system.	Data Inconsistency is less in database management system.
Centralisation is hard to get when it comes to File Management System.	Centralisation is achieved in Database Management System.
User locates the physical address of the files to access data in File Management System.	In Database Management System, user is unaware of physical address where data is stored.
Security is low in File Management System.	Security is high in Database Management System.
File Management System stores unstructured data as isolated data files/entities.	Database Management System stores structured data which have well defined constraints and interrelation.

Ternary Relationship : Suppose we want to keep track of which person got which degree from which University. We could represent this with a ternary relationship as follows :

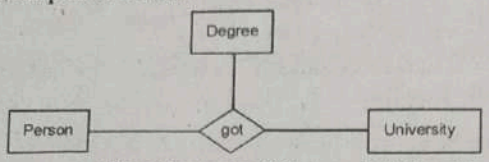


Fig. : Example of Ternary Relationship

Q.14 ***Draw and explain architecture of RDBMS.*** *[R.T.U. 2017]*

Ans. Architecture of RDBMS : There are five major components that are exercised in a typical interaction with an RDBMS:

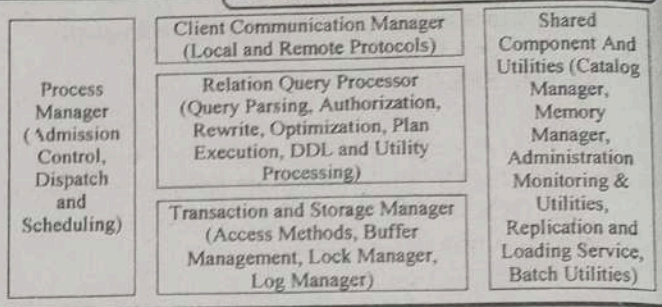
(1) Client Communication Manager : In order to communicate with a database an application needs to make a connection with a database over a network. An application establishes a connection with the Client Communication Manager. This component enables communication between various database clients through both local and remote protocols. Its main responsibility is to remember communication state, return data and control messages (result codes, errors) as well as forward the client's request to other parts of the DBMS.

(2) Process Manager : The process manager is responsible for providing a "thread of computation" for each database request from a database client. It links the threads data and control output to the appropriate communication manager client. The first decision to be made by the process manager is to determine if enough system resources are available to execute the query or defer the same until a later time.

(3) Relational Query Processor : On receiving a request to process a query the relation query processor first checks if the user is authorised to run the query. It then compiles the query into an interim query plan which is further optimised. The resulting plan is executed by the "plan executor" which eventually makes use of the transaction and storage monitor.

(4) Transaction and Storage Manager : Once a query is parsed it retrieves the requested data from the Transaction and Storage Manager. It the gatekeeper to all data access and manipulation calls. The transactional and storage manager also make sure that ACID properties of a transaction are adhered too and thus the need for a lock and log manager.

(5) Shared Components and Utilities : There are a number of shared components and utilities that are essential for a database to run.



Q.15 ***Contrast between DDL and DML.*** *[R.T.U. 2017]*

OR

Differential between DDL and DML using syntax for them. *[R.T.U. 2014]*

OR

Differentiate between DDL and DML. *[R.T.U. 2016]*

Ans. Difference between DDL and DML : 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

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

- SELECT : retrieve data from the 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
- MERGE : UPSERT operation (insert or update)

Transaction control statements manage changes made by DML statements. The transaction control statements are:

COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

All transaction control statements, except certain forms of the COMMIT and ROLLBACK commands, are supported in PL/SQL.

Session control statements dynamically manage the properties of a user session. These statements do not implicitly commit the current transaction.

PL/SQL does not support session control statements. The session control statements are:

ALTER SESSION

SET ROLE

- CALL : call a PL/SQL or Java subprogram
- EXPLAIN PLAN : explain access path to data
- LOCK TABLE : control concurrency

DDL Commands Syntax

The Create Table command

```
CREATE TABLE [schema.]table
   ( { column datatype [DEFAULT expr]
[column_constraint] ... | table_constraint}
   [, { column datatype [DEFAULT expr]
[column_constraint] ... | table_constraint} ]...)
   [AS subquery]
```

The DROP Command

Syntax:

```
DROP TABLE <table_name>
```

Example:

```
DROP TABLE Student;
```

It will destroy the table and all data which will be recorded in it

DML Command Syntax

Viewing data in the table (Select Command) :Once data has been inserted into a table, the next most logical operation would be to view what has been inserted. The SELECT SQL verb is used to achieve this.

All rows and all columns

Syntax: SELECT * FROM Table_name;

eg: Select * from Student; It will show all the table records.

SELECT First_name, DOB FROM STUDENT WHERE Reg_no = 'S101'; Cover it by single inverted comma if its datatype is varchar or char.

This Command will show one row. because you have given condition for only one row and particular records. If condition which has given in WHERE Clause is true then records will be fetched otherwise it will show no records selected.

Q.16 *Discuss types of DBMS.* *[R.T.U. 2017]*

Ans.Types and Classification of Database Management System : On the basis of data model, we have five types of DBMS:

(1) Relational Database : This is the most popular data model used in industries. It is based on the SQL. They are table oriented which means data is stored in different access control tables, each has the key field whose task is to identify each row. The tables or the files with the data are called as relations that help in designating the row or record, and columns are referred to attributes or fields. Few examples are MYSQL(Oracle, open source), Oracle database (Oracle), Microsoft SQL server(Microsoft) and DB2(IBM).

(2) Object Oriented Database : The information here is in the form of the object as used in object oriented programming. It adds the database functionality to object programming languages. It requires less code, use more natural data and also code bases are easy to maintain. Examples are ObjectDB (ObjectDB software).

(3) Object Relational Database : Relational DBMS are evolving continuously and they have been incorporating many concepts developed in object database leading to a new class called extended relational database or object relational database.

(4) Hierarchical Database : In this, the information about the groups of parent or child relationships is present in the records which is similar to the structure of a tree. Here the data follows a series of records, set of values attached to it. They are used in industry on mainframe platforms. Examples are IMS(IBM), Windows registry(Microsoft).

(5) Network Database : Mainly used on a large digital computers. If there are more connections, then this database is efficient. They are similar to hierarchical database, they look like a cobweb or interconnected network of records. Examples are CA-IDMS (COMPUTER associates), IMAGE(HP).

While on the basis of number of users, we have two types of DBMS:

(i) Single User : As the name itself indicates it can support only one user at a time. It is mostly used with the personal computer on which the data resides accessible to a single person. The user may design, maintain and write the database programs.

(ii) Multiple Users : It supports multiple users concurrently. Data can be both integrated and shared,a database should be integrated when the same information is not need be recorded in two places. For example a student in the college should have the database containing his information. It must be accessible to all the departments related to him. For example the library department and the fee section department should have information about student's database. So in such case, we can integrate and even though database resides in only one place both the departments will have the access to it.

Q.17 ***Explain advantage of DBMS over file system.*** *[R.T.U. 2017]*

OR

What do you mean by DBMS? Explain the advantages of Database management system over file management system. *[R.T.U. 2016]*

OR

Explain the advantages of Database Management System over File Management System. *[R.T.U. 2015]*

OR

List out four significant differences between a file processing system and a DBMS. *[Raj. Univ. 2007]*

OR

Advantage of DBMS over File Processing System. *[R.T.U. 2010]*

Ans. DBMS : *Refer to Q.5.*

Advantages of Database Management System Over File Management System :

1. As we do not have 1000 GB main memory (primary memory) to store the data, so we store the data in some permanent storage device (secondary memory) like magnetic disk or magnetic tape etc. Hence, file oriented system fails in primary memory cases and we apply database management system to store the data files permanently.
2. Suppose if we have a large amount of primary memory on a 16 bit or 32 bit computer system, then a problem can occur in file based system to use the data by direct or random addressing. Also we cannot call more than 2GB or 4GB of data direct to the primary memory at a time. So, we need a database program to identify the data.
3. Some programs are too lengthy and complex due to which they cannot store large amount of data in the files related to the operating systems. But a database system makes it simple and fast.
4. We cannot change and access file-oriented data simultaneously, so we have requirement of such type of system (DBMS) which can be used to access the large amount of data concurrently.
5. Also we cannot recall or recover the file oriented data, but centralized database management can solve such type of problem in DBMS.
6. File oriented operating system provide only a password mechanism for security, but this is not successful in case of number of users are accessing the same data by using the same login.

At the end, we can say that DBMS is a piece of software that is designed to make the processing faster and easier.

Q.18 ***Why query processor component of database system is important? Briefly discuss about all components of query processor.*** *[R.T.U. 2016, 2014]*

Ans. Query Processor Component : The query processor components are important because it helps the database system simplify and facilitate access to data. High level views help to achieve this goal; with them, users of the system are not burdened unnecessarily with the physical details of the implementation of the system. However, quick processing of updates and queries is important. It is the job of the database system to translate updates and queries is important. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

These components are used in evaluating DDL and DML queries and includes following components.

(i) DML Compiler : It first attemps to transform user's request into an equivalent but more efficient form and then translates that into a set of low level instructions that can be used by query evaluation engine.

(ii) Embedded DML Pre-Compiler : It Converts DML statements embedded in an application program to normal procedure calls in host language. This pre compiler consult DML compiler to generate the appropriate code.

(iii) DDL Interpreter : It makes data dictionary, which contains metadata.

(iv) Query Evaluation Engine : It executes low instructions generated by the DML compiler.

Q.19 ***Briefly discuss the history of database systems.*** *[R.T.U. 2015]*

Ans. Database Systems : The history of database management systems begins around the time when computers began taking off. In 1960's, the concept of the database was put in use and also began to grow in commercial use and it was this rise that was an interest to a man named Charles W. Bachman. He invented the database management system. Charles Bachman was an industrial researcher, beginning a career at Dow Chemical, where he became the data processing manager in 1950's before leaving to work at General Electric in 1960.

It was in 1960 that Bachman came up with the Integrated Database System, the very first DBMS. This got the proverbial ball rolling, as Bachman found the Database Talk Group along with the group that gave the

standardization to the programming language of COBOL. Not to be left out, IBM created their own database system, known IMS, for that of NASA's Apollo space program. Both of these are now known as the precursors of navigational databases.

The earlier invented DBMS was not easy to use. Thus, in 1970's a man named Edgar Codd thought of a way to make things a bit easier. Codd worked for IBM and felt that there had to be a way to make things easier when using these DBMS. He wrote a paper entitled, A Relational Model of Data for Large Shared Data Banks, in which he proposed replacing these current systems with that of tables and rows. This concept became relational DBMS.

IBM started working on a prototype system loosely based on Codd's concepts in the early 1970's. The first version was ready in 1974/5, and then started work on multi-table systems in which the data could be split so that all of the data for a record (some of which is optional) did not have to be stored in a single large "chunk". Subsequent multi-user versions were tested by customers in 1978 and 1979, and by the time a standardized query language (SQL) had been added.

Then in 1990's, object-oriented programming was on its peak, which gets reflected in databases also bringing in picture Object Databases and Object-Relational Databases.

Q.20 ***Explain the notational conventions used in the ER model.*** ***[R.T.U. 2015]***

Ans. All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. The symbols used for the basic ER constructs are:

- Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.
- Attributes are represented by ellipses.
- A solid line connecting two entities represents relationships. The name of the relationship is written above the line. Relationship names should be verbs and diamond sign is used to represent relationship sets.
- Attributes, when included, are listed inside the entity rectangle. Attributes, which are identifiers, are underlined. Attribute names should be singular nouns.
- Multi-valued attributes are represented by double ellipses.
- Directed line is used to indicate one occurrence and undirected line is used to indicate many occurrences in a relation.

The symbols used to design an E-R diagram is shown below.

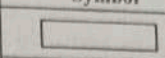
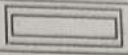
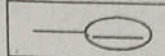
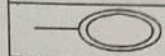
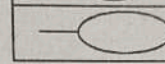
Symbol	Meaning
	Entity Type
	Weak Entity Type
	Relationship Type
	Identifying Relationship Type
	Attribute
	Key Attribute
	Multivaluated Attribute
	Composite Attribute
	Derived Attribute

Fig.

Q.21 ***Describe the following terms with examples:***
(i) Owner entity sets
(ii) Identifying relationship
(iii) Discriminator
and design E–R diagram to model them. ***[R.T.U. 2014]***

Ans. (i) Owner Entity Sets

(i) The identifying relationship will be one-to-many from owner entity set to weak entity set.

(ii) The participation of owner entity set in the identifying relationship will be partial and the participation of the weak entity set in the identifying relationship will be total.

As shown in fig.1 set E_2 is said to be existence-dependent on E_1 and E_1 is said to be the owner entity set of E_2. The relationship R between E_2 and E_1 is called identifying relationship. The weak entity set E_2 will have a set of attributes called its discriminator, which together with the primary key of E_1 will form the primary key of E_2.

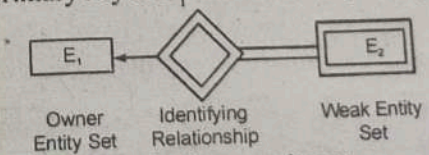


Fig.1 : Owner entity sets

(ii) Identifying Relationships

For a weak entity set to be meaningful, it must be associated with another entity set, called the identifying or owner entity set. Every weak entity must be associated

with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying entity set. The identifying entity set is said to own the weak entity set that is identifies. The relationship associating the weak entity set with the identifying entity set is called the identifying relationship.

The identifying relationship is many-to-one from the weak entity set to the identifying entity set, and the participation of the weak entity set in the relationship is total . The identifying relationship set should not have any descriptive attributes, since any such attributes can instead be associated with the weak entity set.

Consider the relationship between APARTMENT and BUILDING as shown in fig.2. In reality, two apartments built in two buildings may have the same apartment number. To distinguish between these two apartments, you may need to attach the building number to the apartment number. In such a case, we can say that the apartment id depends on the existence of the building id. That is, the identifier for the APARTMENT entity should include the identifier for the BUILDING entity. Such a relationship is called the identifying relationships.

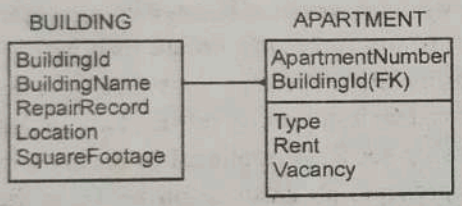


Fig. 2 : Identifying relationship

The identifying relationship is represented by a solid line. The parent entity is represented by a square and the id dependent entity is represented by a round-cornered square.

(iii) Discriminator : The weak entity set is dependent on a strong entity set and has a discriminator.

The discriminator of a weak entity set is an attribute or a set of attributes whose value uniquely identifies each weak entity from a set of weak entities related to one strong entity whose primary key value is given. For example, the discriminator of the weak entity set departures (fig.3) is the attribute date, since for each flight the date values are different.

The primary key of the weak entity set is formed by the primary key of the strong entity set plus its discriminator.

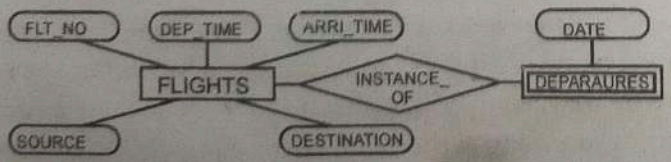


Fig.3 : E-R diagram for a weak entity set which is dependent on a strong entity set

Example : Suppose an entity set EMPLOYEE (EMP_ID, EMP_NAME, SALARY, DEPENDENTS) has an attribute DEPENDENT which is multi-valued i.e. has an employee may have none or one or more than an employee may have none or one or more than dependents. This situation can be best modeled as follows:

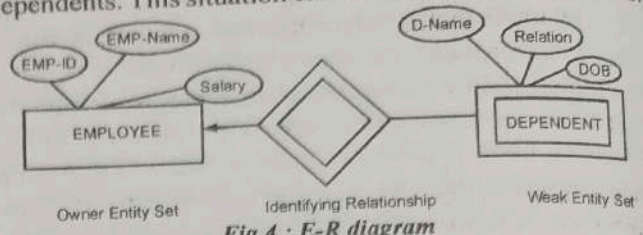


Fig.4 : E-R diagram

- The weak entity set DEPENDENT is existence dependent on the strong entity set EMPLOYEE.
- The weak entity set "DEPENDENT' has a discriminator attribute D_NAME, which along with primary key EMP_ID of EMPLOYEE, forms primary key of the weak entity set DEPENDENT. In E-R diagram, the Discriminator (also called partial key) of a weak entity set is marked by underlining with a broken line.

Q.22 ***What is view in DBMS? List two reasons why we may choose to define a view and list two major problems with processing update operations expressed in terms of views.*** *[R.T.U. 2013]*

Ans. View Level : It is the highest level of abstraction which describes different views of the entire database. These views are designed according to the requirements of user who wants to access only a part of the database. A database may have several views, according to the demand of individual user or the group of users. The data in these views are not exactly stored in DBMS but they are computed using specification of view described by user. An analogy to the concept of data types in programming language may clarify the distinction among levels of abstraction. Most high-level programming languages support the notion of a record type.

At the view level, computer users see a set of application programs that hide details of the data type. Similarly, at the view level, several views of the database are defined, and database users see these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing parts of the database.

Customer_Loan			View
Cust_ID	:	101	
Loan_No	:	1011	
Amount_in_Dollars	:	8755.00	

Two reasons why we may choose to define a view :

(a) Security conditions may require that the entire logical database should not visible to all users.

(b) We may wish to create a personalized collection of relations that is better matched to a certain user's intuition than is the actual logical model.

Two major problems with processing update operations expressed in terms of views : Views present significant problems if updates are expressed with them. The difficulty is that a modification to the database expressed in terms of a view must be translated to a modification to the actual relations in the logical model of the database :

(a) Since the view may not have all the attributes of the underlying tables, insertion of a tuple into the view will insert tuples into the underlying tables, with those attributes not participating in the view getting null values. This may not be desirable, especially if the attribute in question is part of the primary key of the table.

(b) If a view is a join of several underlying tables and an insertion results tuples with nulls in the join columns, the desired effect of the insertion will not be achieved. In other words, an update to a view may not be expressible at all as updates to base relations.

Q.23 ***Write short notes on :***

(i) Participation constraints.

(ii) Role in ER-data model

(iii) Identifying Relationship. ***[R.T.U. 2013]***

Ans.(i) Participation Constraints : The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R. If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial. For example, we expect every loan entity to be related to at least one customer through the borrower relationship. Therefore the participation of loan in the relationship set borrower is total. In contrast, an individual can be a bank customer whether or not she has a loan with the bank. Hence, it is possible that only some of the customer entities are related to the loan entity set through the borrower relationship, and the participation of customer in the borrower relationship set is therefore partial.

Ans.(ii) The Entity-Relationship Model : *Refer to Q.1.*

The E-R data model employs three basic notations entity sets, relationship sets and attributes.

Entity Sets : An entity is a " thing" or "object" in the real world that is distinguishable from all other objects.

For example each person in an enterprise is an entity. An entity has a set of properties and the values for some set of properties may uniquely identify an entity.

An entity set is a set of entities of the same type that share the same properties or attributes. Entity sets do not need to be digitised. An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.

Each entity has a value for each of its attributes.

A database thus includes a collection of entity sets, each of which contains any number of entities of the same type.

Relationship Sets : A relationship is an association among several entities.

A relationship set is a set of relationships of the same type. Finally, it is a mathematical relation on $n \geq 2$ entity sets. If $E_1, E_2, \ldots E_n$ are entity sets, then relationship set R is a subset of

$$\{(e_1, e_2, \ldots e_n)/e, e_1 \in E_1, e_2 \in E_2 \ldots, e_n \in E_n\}$$

where $(e_1, e_2 \ldots e_n)$ is a relationship.

The association between entity sets is referred to as participation, that is, the entity sets $E_1, E_2 \ldots E_n$ participate in relationship set R. A relationship instance in an E-R schema represents an association between the named entities in the real world enterprise.

A relationship may also have attributes called **descriptive** attributes.

Attributes : For each attribute, there is a set of permitted values, called the domain or value set of that attribute.

Formally, an attribute of an entity set is a function that maps from the entity set into domain. Since an entity set may have several attributes, each entity can be described by a set of (attribute, data value) pairs, one pair for each attribute of the entity set.

An attribute, as used in the E-R model, can be characterized by the following attribute types :

(a) Simple and composite attributes

(b) Single-valued and multi-valued attributes

(c) Derived attribute

An E-R diagram can express the overall logical structure of a database graphically. E-R diagrams are simple and clear qualities that may well account in large part for the widespread use of the E-R model.

Ans.(iii) Identifying Relationship : *Refer to Q.21(ii).*

PART-C

Q.24 (a) Explain the aggregation v/s ternary relationship in detail. *[R.T.U. 2019]*

OR

Differentiate between Ternary relationship and aggregation. *[R.T.U. 2015, Dec.2013]*

(b) Explain the entity v/s attribute in detail. *[R.T.U. 2019]*

OR

Differentiate between entity and attribute. *[R.T.U. 2015, Dec.2013]*

Ans.(a) Difference between Ternary Relationship and Aggregation :

Ternary Relationship : *Refer to Q.13.*

Aggregation : Aggregation used when we have to model a relationship involving another relationship set.

Aggregation allows us to treat a relationship set as an entity set for purposes of participating in (other) relationships.

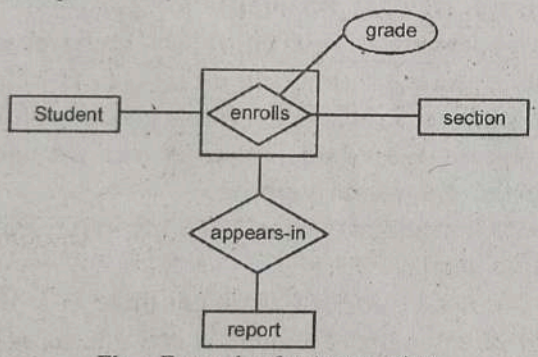


Fig. : Example of a Aggregation

Ans.(b) Difference between Entity and Attributes :

Entity : *Refer to Q.4.*

Attributes : Entities are described in a database by a set of properties. These properties are referred as attributes.

For example, the attributes account-number and balance may describe one particular account in a bank and they form attributes of the account entity set. Similarly, attributes customer-name, customer-street address and customer-city may describe a customer entity.

Q.25 ***Explain inventory management system.*** *[R.T.U. 2017]*

Ans. Inventory Management System : The Inventory Management System is a real-time inventory database capable of connecting multiple stores. This can be used to track the inventory of a single store, or to manage the distribution of stock between several branches of a larger franchise. However, the system merely records sales and restocking data and provides notification of low stock at any location through email at a specified interval. The goal is to reduce the strain of tracking rather than to handle all store maintenance. Further features may include the ability to generate reports of sales, but again the interpretation is left to the management. In addition, since theft does occasionally occur, the system provides solutions for confirming the store inventory and for correcting stock quantities.

Typically, a company maintains one or both types of inventory; stock items and non-stock items. Stock items are stored products or parts that are ready for sale. Non-stock items are items that are used by the company, such as office supplies.

You can use an Inventory Management System to:

- Identify, store, and track stock and non-stock items.
- Identify and track prices in multiple currencies.
- Identify and store items that require special handling such as refrigeration.
- Identify items that require quality analysis or testing.
- Determine obsolete items.
- Identify and account for broken or defective parts.

A typical Inventory Management System may provide following business processes:

(1) Item classification : Provides for numerous purchasing, sales, and distribution classifications to report on purchasing or sales activity using many different facets of item characteristics and to determine how products move through or reside within the warehouse.

(2) Unit of measure conversions : Enables to define package size and the relationships among packages and performs standard conversions, such as pounds to ounces or count to dozens.

(3) Dual units of measure : Enables to maintain inventory and perform transactions for items in two units of measure.

(4) Manufacturing information : Enables to define the elements of items to enhance inventory planning and lead time forecasting. As companies move toward leaner inventories, such accurately forecasting is critical to successful operations.

(5) Item grade and potency information : Enables to monitor grade and potency, which is crucial in

industries such as food and drug manufacturing. In many cases, recording and tracking processes are strictly regulated, and noncompliance can result in stiff penalties. Furthermore, regulatory agencies require extensive documentation.

(6) Issues : Issues are typically used to remove inventory from a location. You can enter an issue for damaged goods, marketing demonstration, or internal use.

(7) Adjustments : Adjustments are used to reconcile discrepancies between physical inventory counts and on-hand system quantities. You can enter an adjustment for shrinkage, unrecorded gain, and initial balances.

(8) Transfers : Transfers are used to document the movement of an item. You can enter a transfer to record movement from location to location, from vehicle to location, or from plant to plant.

(9) Physical inventories : Physical inventories provide cycle count and tag count to conduct periodic physical inventory reconciliations.

The Inventory Management System's Database may contain the following tables:

(i) Basic information about each warehouse location, such as zones and level of detail.

(ii) Basic information about each item. For example, item number, description, search keys, category codes, default units of measure, etc.

(iii) Default item information. For example, each item's process and dimension groups, other parameters common to every unit of that item in the warehouse.

(iv) Inventory cost records.

(v) History of all inventory movements.

(vi) Predefined messages that print on documents such as sales orders and purchase orders.

(vii) Order types (sales, procurement, and so on) and the order statuses at which the system creates a request.

Q.26 ***Draw ER diagram of any one of the following and explain each component of this ER diagram. Library management system.*** ***[R.T.U. 2017]***

OR

What is the role of ER model in database design? Draw an ER diagram for library management system and convert ER-diagram into tables.

Ans. *Refer to Q.12.*

Q.27 ***Draw the diagram of system structure of DBMS. Write down the main function of each component.*** ***[R.T.U. 2016]***

Ans. Database System Structure : A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

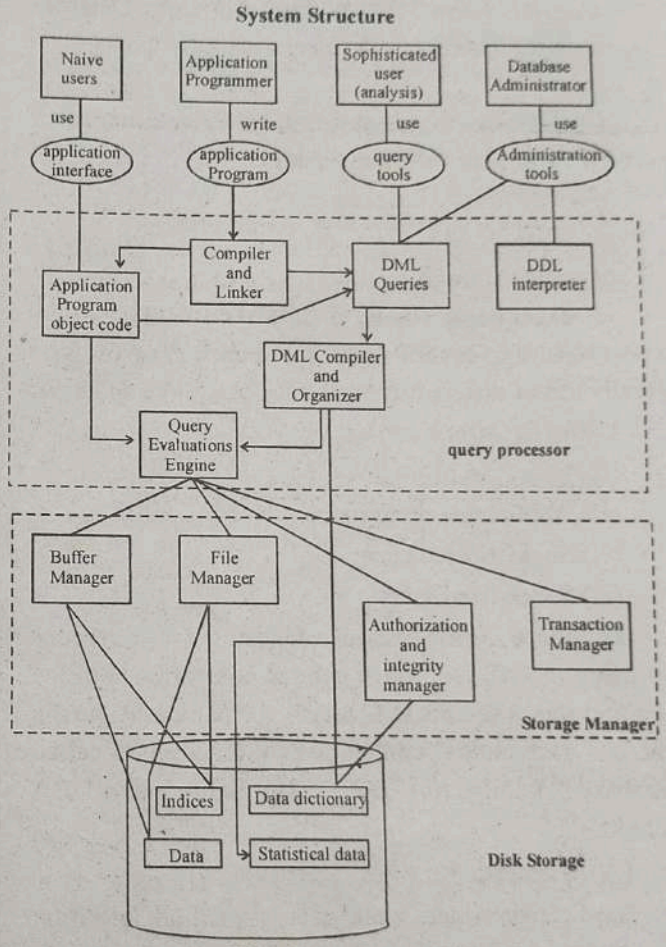


Fig. : System Structure

The storage manager is important because databases typically require a large amount of storage space. Separate databases range in size from hundreds of gigabytes to, for the largest database, terabytes of data. A gigabyte is 1000 mega bytes (1 billion bytes) and a terabyte is 1 million mega bytes (1 trillion bytes). Since main memory of computer cannot store this much information, the information is stored on disks. Data is moved between disk storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that

the database system structures the data so as to minimize the need to move data between disk and main memory.

The *query processor* is important because it helps the database system to simplify and facilitate access to data. High level views helps to achieve this goal; with them, users of the system are not bounding unnecessarily with the physical details for the implementation of the database system to translate updates and queries written in a non-procedural language, at the logical level, into an efficient sequence of operations at the physical level.

Components of DBMS Environment : Database Management System environment's main components are as :

(i) Database Users and Database Administrator
(ii) Transaction Management
(iii) The Query Processor
(iv) Storage Manager
(v) Disk Storage

(i) Database Users : Database users can be categorized into several classes and each class of users usually uses a different type of interface to the database.

Different users are as :

(A) Naive users
(B) Application programmer
(C) Sophisticated users
(D) Specialized users

Database Administrator : One of the main reasons for using DBMS is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called Database Administrator (DBA). The functions of DBA includes :

(A) Schema definition
(B) Storage structure and access- method definition
(C) Schema and Physical –Organization modification
(D) Granting of authorization for data access
(E) Routine maintenance

(ii) Transaction Management : The transaction management subsystem is responsible for ensuring that the database remains in a consistent (correct) state despite system failures. The transaction manager also ensures that concurrent transaction execution proceed without conflicting.

(iii) The Query Processor : The query processor subsystem compiles and executes DDL and DML statements.

The query processor components include :

(A) DDL interpreter
(B) DML compiler
(C) Query evaluation engine

(iv) Storage Manager : The storage manager subsystem provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.

The storage manager components include :

(A) Authorization and integrity manager
(B) Transaction manager
(C) File manager
(D) Buffer manager

(v) Disk Storage : The storage manager implements several data structures as part of the physical system implementation.

(A) Date files
(B) Data dictionary
(C) Indices

Q.28 ***Differentiate specialization and generalization.*** *[R.T.U. 2010, 2008]*

OR

Explain concept of specialization and generalization in E-R model. *[R.T.U. 2016]*

OR

Write short note on specialization and generalization in E-R model. *[Raj. Univ. 2007, 2006, 2000]*

Ans. Although the basic E-R concept can model most database features, some aspect of a database may be more optly expressed by certain extensions Specialization and Generalization.

Specialization : An entity set may include subgrouping of entities that are distinct in some way from the atttributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinct entity grouping.

Consider an entity set person with attributes name, street and city. A person may be further classified as one of the following :

- Customer
- Employee

Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibility additional attributes. For example- customer entities may be described further by the attributes customer-id whereas employee entities may be described further by the attributes employee-id and salary. The process of designing subgrouping within an entity set is called *specialization*. The specialization of person allows us to distinguish among person according to whether they are employee or customers.

An another example, suppose the bank wishes to divide accounts need two categories, checking account and saving account. Saving accounts need a minimum balance, but the bank may set interest rates differently for different customers, offering better rates favored customers. Checking account must be recorded.

The bank could then create two specialization of account, namely saving-account and checking-account. The entity set saving-account would have all the attributes account and an additional attribute overdraft account.

We can apply specialization repeatedly to refine a design schema for instance bank employees. It may be further classified as one of the following :

- officer
- teller
- secretry

Each of these employee types is described by a set of attributes that includes all the attributes of entity set employee plus additional attributes. For example- officer entites may be described further by the attributes office-number and hours-per-week and secretary entities by the attributes hours-per week further secretary entities may participate in a relationship secretary for which identifiers and employees are assisted by a secretary.

An entity set may be specialized by more than one distinguishing feature. In our example the distinguishing feature among employee entities is the job the employee performs. Another coexistent specialization could be based on whether the person is a temporary- employee and permanent- employee when more than one specialization is formed on an entity set, a particular entity may belong to multiple who is a secretary.

In terms of an E-R diagram, specialization is depicted by a triangle component labeled IS A, as figure shows. The label IS A stands for "is a" for example that a customer "is a" person. The IS A relationship may also be referred to as a superclass-subclass relationship. Higher and lower level entity sets are depicted as regular entity sets i.e. as rectangles containing the name of the entity set.

Generalization : It is simple inversion of specialization. The refinement from an initial entity set into successive level of entity subgroupings represent a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into higher-level entity set on the basis of common features. The database designer may have first identified a customer entity set with the attributes like name, street, city and customer-id and an employee entity set with the attributes like name, street, city, employee-id and salary.

There are similarities between the customers entity set and the employee entity set in the sense that they have several attributes in common. This commonality can be expressed by generalization, which is a containment relationship that exists between a higher-level entity set and one or more lower-level entity sets. Higher and lower-level entity sets also may be designed by the terms superclass and subclass, respectively the person entity set is the superclass of the customer and employee subclasses.

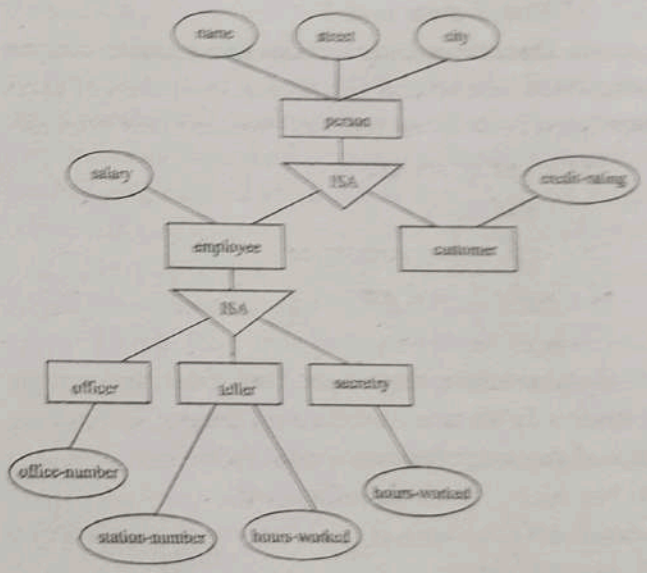


Fig. : Specialization and generalization

For all practical purpose generalization is a simple inversion of specialization we will apply both process, in combination in the course of designing the E–R schema for an enterprise. In terms of the E-R diagram itself we do not distinguish between specialization and generalization new level of entity representation will be distinguished or synthesized as the design schema comes to express fully the database application and the requirement of the database. Difference in the two approaches may be characterized by their starting point and overall goal.

Specialization stems from a single entity set; it emphasizes differences among entities within the set by creating distinct lower-level entity sets. These lower-level entity sets may have attributes or may participate in the

relationship they do not apply to all the entities in the higher level entity set. Indeed the reason a designer applies specialization is to represent such distinctive features. If customer and employee neither have attributes that person entities do not have nor participate in different relationship than those in which person entities participate there would be no need to specialization the person entity set.

Generalization proceeds from the recognition that a number of entity set share some common features. On the basis of their commonalities generalization synthesizes these entity sets into a single higher level set.

Q.29 ***Explain network and object oriented model. What are the roles of these models in database design?*** ***[R.T.U.2015, Dec.2013]***

OR

Define data model. Explain similarities and differences among network, hierarchical and relational data model. ***[R.T.U. 2013]***

Ans. Data Model : Data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. Generally two models are used for database design Relational model and Entity-Relationship model. Both provide a way to describe the design of a database at the logical level.

1. Relational Model : The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns and each column has a unique name. Given table represents a simple relational database:

Table 1 : Customer

Customer-Id	*Customer-Name*	*Customer-Street*	*Customer-City*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

Table 2 : Account

Account-Number	*Balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

Table 3 : Depositor

Customer-Id	*Account-Number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

Table 1. shows details of bank customers, table 2 shows accounts and table 3 shows which accounts belong to which customers.

The first table, the *customer* table, shows, for example- that the customer identified by customer-id 192-83-7465 is named Johnson and lives at 12 Alma St. in Palo Alto.

The table 2, *account*, shows, for example- that account A-101 has a balance of $500 and A-201 has a balance of $900.

The table 3 shows which accounts belong to which customers. For example- account number A-101 belongs to the customer whose customer-id is 192-83-7465, namely Johnson and customers 192-83-7465 (Johnson) and 019-28-3746 (Smith) share account number A-201 (they may share a business venture).

The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields or attributes. The columns of the table correspond to the attributes of the record type.

It is not hard to see how tables may be stored in files. For instance, a special character (such as a comma) may be used to delimit the different attributes of a record and another special character (such as a newline character) may be used to delimit records. The relational model hides such low-level implementation details from database developers and users.

The relational data model is the most widely used data model and a vast majority of current database systems are based on the relational model.

The relational model is at a lower level of abstraction than the E-R model. Database designs are often carried out in the E-R model and then translated to the relational

model. For example, it is easy to see that the tables *customer* and *account* correspond to the entity sets of the same name, while the table *depositor* corresponds to the relationship set *depositor*. We also note that it is possible to create schemas in the relational model that have problems such as unnecessarily duplicated information. For example, suppose we store *account-number* as an attribute of the *customer* record. Then, to represent the fact that accounts A-101 and A-201 both belong to customer Johnson (with customer-id 192-83-7465), we would need to store two rows in the *customer* table. The values for *customer-name*, *customer-street* and *customer-city* for Johnson would get unnecessarily duplicated in the two rows.

2. Other Data Models : The **Object-Oriented data model** is another data model that has seen increasing attention. The object-oriented model can be seen as extending the E-R model with notions of encapsulation, methods and object identity.

Historically, two other data models, **the network data model** and the **hierarchical data model,** preceded the relation of data model. These models were tied closely to the underlying implementation and complicated the task of modeling data.

Semi-structured data models permit the specification of data where indivisible data items of the same type may have different sets of attributes. This is in contrast with the data models, where every data item of a particular type must have the same set of attributes. The extensible mark up language (XML) is widely used to represent semi-structured data.

Similarities and Differences Among Different Data Models :

S. No.	Hierarchical Data Model	Network Data Model	Relational Data Model
1.	Relationship between records is of the parent child type (Trees).	Relationship between records is expressed in the form of pointers or links (Graphs).	Relationship between records is represented by a relation that contains a key for each record involved in the relationship.
2.	Many to many relationship cannot be expressed in this model.	Many to many relationship can also be implemented.	Many to many relationship can be easily implemented.
3.	It is a simple, straight forward and natural method of implementing record relationships.	Record relationship implementation is very complex due to the use of pointers.	Relationship implementation is very easy through the use of a key or composite key field(s).
4.	This type of model is useful only when there is some hierarchical character in the database.	Network model is useful for representing such records which have many to many relationships.	Relational model is useful for representing most of the real world objects and relationship among them.
5.	In order to represent links among records, pointers are used. Thus relations among records are physical.	In network model also, the record relations are physical.	Relational model does not maintain physical connection among records. Data is organized logically in the form of rows and columns and stored in table.
6.	Searching for a record is very difficult since one can retrieve a child only after going through its parent record.	Searching a record is easy since there are multiple access paths to data elements.	A unique indexed key field is used to search for a data element.

Similarities and Differences Based on the Operations Performed :

Operation	Hierarchical Model	Network Model	Relational Model
1. Insert operation	We cannot insert the information of a child who does not have any parent.	It does not suffer from any insert anomaly.	It does not suffer from any insert anomaly.
2. Update operation	There are multiple occurrences of child records, which lead to problem of inconsistency during the update operation.	This model is free from update anomalies because there is only a single occurrence for each record set.	This model is also free from update anomalies because it removes the redundancy of data by proper designing through normalization process.

3. Delete operation	It is based on parent child relationship and deletion of parent results in deletion of child records also. This is known as a cascading effect.	It is based on many-to-many (M:N) relationship which make it free from delete anomalies.	It is also free from delete anomalies as the information is stored in different tables.
4. Retrieve operation	Retrieve algorithms are complex and asymmetric.	Retrieve algorithms are complex and symmetric.	Retrieve algorithms are simple and symmetric.

Network Model and its role in Database Design : The network data model preseded the relation al data model. This model was tied closely to the underlying implementation and complicated the task of data modelling.

(i) Relationship between records is expressed in the form of pointer or links (graphs).

(ii) Many-to-many relationship can also be implemented. *"Network model is useful for representing such records which have m-to-m relationships."*

(iii) Searching a record is easy since these are multiple access paths to a data element.

(iv) It does not suffer from any insert anomaly.

(v) This model is free from update and delete anomalies.

Object Oriented Model and its Role in Database Design : The object-oriented data model extends the representation of entities by adding notions of encapsulation, methods (functions), and object identity. The object-oriented model is based on the OOP paradigm and overcome the relational model's restrictions.

- The object-oriented data model has been developed to deal with several new type of applications.
- It is based on the concept of encapsulating in an object, the data and the code that operates on those data.

The main concepts of the object-oriented data model are as follows :

(i) Object : An object corresponds to an entity in the E-R model. In general, an object has associated with it.

- A set of variables
- A set of messages
- A set of methods

(ii) Class : The group of similar objects is called class. Examples of classes are employees, customers, accounts, and loan etc.

(iii) Inheritance : The concept of a class hierarchy is similar to the concept of specialization hierarchy in the entity relationship model. For example, employees and customers can be represented by classes that are specializations of a person class.

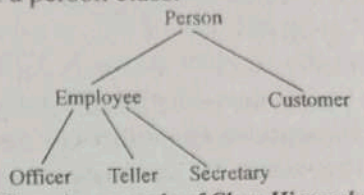


Fig. : An example of Class Hierarchy

Q.30 ***(a) Compare the file system and DBMS on the basis of following :***

(i) Integrity

(ii) Difficulty in accessing data

(iii) Concurrently access anomalies

(b) Describe various functions of Database Administrator.

(c) Explain the database design process.

(d) Using neat diagram and examples show that database system hides details of data stored and maintained. ***[R.T.U. 2014]***

Ans. (a) (i) Integrity : The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount. These constraints are enforced in the system by adding appropriate code in the various application programs. When new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items for different files.

Integrity of data means that data in database is always accurate, such that incorrect information cannot be stored in database. In order to maintain the integrity of data, some integrity constraints are enforced on the database. A DBMS should provide capabilities for defining and enforcing the constraints. So, we can conclude that integrity constraint can be easily enforced in centralized DBMS system as compared to file system.

(ii) Difficulty in Accessing Data: The conventional file processing environments do not allow needed data to be retrieved in a convenient and efficient manner. Better data retrieval system must be developed for general use. Consider the airline reservation system. If the senior management of company wants to access the information

of all its customers who are living in the same postal code, it has to be done manually because current file processing system does not allow the user to obtain this information. So in the above case, there are two options. Either the application programmer has to write a new application program to satisfy the unusual request or could get this information manually .In former case, it doesn't guarantee that the same query will be asked and same application program would be used in future.

If a query changes, a new application program should be written to get the needed information.

The database system utilizes different sophisticated techniques to access the stored data very efficiently. The database systems provide query languages or report writers that allow the users to ask adhoc queries to obtain the needed information immediately, without the requirement to write application programs (as in case of file system), that access the information from the database. This is possible due to integration in database systems.

(iii) Concurrent Access Anomalies : In order to speed up the performance of the system and faster response to applications, many systems allow the user to update the data concurrently. Suppose two users located at different locations wants to book the tickets, there might be situation that both of the people will be given the same seat because the data is stored in multiple locations and both of them will be given a seat from individual copy of the data.

Therefore there should be some protection mechanisms to avoid this concurrent updates. DBMS systems provide mechanisms to provide concurrent access of data to multiple users. Database can manage concurrent data access effectively. It ensures no interference between users that would not result any loss of information or loss of integrity.

Ans. (b) Database Administrator (DBA) : One of the main reasons for using DBMS is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a Database Administrator (DBA).

The functions of a DBA

(i) **Schema definition :** The DBA creates the original database schema by executing a set of data definition statements in the DDL.

(ii) Storage structure and access-method definition.

(iii) **Schema and physical-organization modification :** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization or to alter the physical organization to improve performance.

(iv) **Granting of authorization for data access :** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

(v) **Routine maintenance :** Examples of the database administrator's routine maintenance activities are :

- Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
- Ensuring that enough free disk space is available for normal operations and upgrading disk space as required.
- Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

Ans. (c) Database Design Process : Database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data defination language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity.

Design Process

1. **Determine the Purpose of Database :** This helps prepare for the remaining steps.
2. **Find and Organize the Information Required:** Gather all of the types of information to record in the database, such as product name and order number.
3. **Divide the Information into Tables :** Divide information items into major entities or subjects, such as products or orders. Each subject then becomes a table.
4. **Turn Information Item into Columns :** Decide what information needs to be stored in each table. Each item becomes a field, and is displayed as a column in the table. For example, an employees table might include field such as last name and hire date.
5. **Specify Primary Keys :** Choose each table's primary key. The primary key is a column, or a set of columns, that is used to uniquely identify each row. An example might be product ID or order ID.

6. **Set up the Table Relationships :** Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. **Refine the Design :** Analyze the design for errors. Create tables and add a few records of sample data. Check if results come from the tables as expected. Make adjustments to the design, as needed.
8. **Apply the Normalization Rules :** Apply the data normalization rules to see if tables are structured correctly. Make adjustments to the tables.

Ans. (d) A major purpose of a database system is to provide users with an **abstract view** of the data *i.e.* the system hides certain details is of how the data are stored and maintained.

Levels of Abstraction : For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify user's interactions with the system.

1. Physical level : The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

2. Logical level : The next-higher level of abstraction describes what data are stored in the database and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

3. View level : The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

For example, in a Pascal-like language, we may declare a record as follows:

```
type customer = record
                    customer_id: string;
                    customer_name: string;
                    customer_street: string;
                    customer_city: string;
                    end;
```

This code defines a new record type called *customer* with four fields. Each field has a name and a type associated with it. A banking enterprise may have several such record types, including

(a) *account*, with fields *account-number* and *balance*

(b) *employee*, with fields *employee-name* and *salary*

At the **physical level,** a customer, account, or employee record can be described as a block of consecutive storage locations (for example, words or bytes).

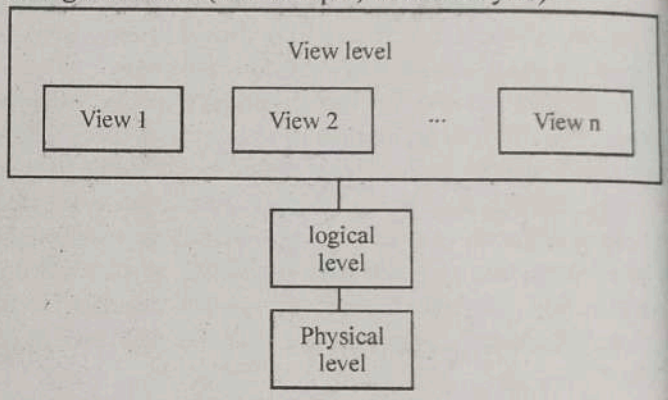


Fig. : The three levels of data abstraction.

The language compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the **logical level**, each such record is described by a type definition, as in the previous code segment and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the **view level**, computer users see a set of application programs that hide details of the data types. Similarly, at the view level, several views of the database are defined and database users see these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database. For example, tellers in a bank see only that part of the database that has information on customer accounts; they cannot access information about salaries of employees.

DataBase Schemas : A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time corresponds to an instance of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema and can usually be changed easily without affecting application programs. Application programs are said to exhibit physical data independence if they do not depend on the physical schema and thus need not be rewritten if the physical schema changes.

Q.31 ***Consider following transaction that transfer $50 from account A to account B.***

Ti : read (A) : reading A from database
A = A – 50 :
write (A) : updating A in database
read (B) :
B = B + 50 :
write (B) :
using the above example describe the following problems in file system :
(i) Atomicity
(ii) Inconsistency ***[R.T.U. 2014]***

Ans. (i) Atomicity Problem: A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account A but was not credited to account B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic-it must happen in its entirely or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

Suppose that, just before the execution of transaction T_i, the values of accounts A and B are $1000 and $2000, respectively. Now suppose that, during the execution of transaction T_i, a failure occurs that prevents T_i from completing its execution successfully. Further, suppose that the failure happened after the write (A) operation but before the write(B) operation. In this case, the values of accounts A and B reflected in the database are $950 and $2000. The system destroyed $50 as a result of this failure. In particular, we note that the sum A + B is no longer preserved.

Thus, because of the failure, the state of the system no longer reflects a real state of the world that the database is supposed to capture. We term such a state an inconsistent state. We must ensure that such inconsistencies are not visible in a database system. Note, however, that the system must at some point be in an inconsistent state. Even if transaction T_i is executed to completion, there exists a point at which the value of account A is $950 and the value of account B is $2000, which is clearly an inconsistent state. This state, however, is eventually replaced by the consistent state where the value of account A is $950, and the value of account B is $2050. Thus, if the transaction never started or was guaranteed to complete, such an inconsistent state would not be visible except during the execution of the transaction. That is the reason for the atomicity requirement. If the atomicity property is present, all actions of the transaction are reflected in the database, or none are.

The basic idea behind ensuring atomicity is this: The database system keeps track (on disk) of the old values of any data on which a transaction performs a write. This information is written to a file called the log. If the transaction does not complete its execution, the database system restores the old values from the log to make it appear as though the transaction never executed. Ensuring atomicity is the responsibility of the database system; specifically, it is handled by a component of the database called the recovery system.

(ii) Inconsistency : Data inconsistency means different copies of the same data are not matching. That means different versions of same basic data are existing. This occurs as the result of update operations that are not updating the same data stored at different places.

Example : Address information of a customer is recorded differently in different files . Same data which has been repeated at several places may not match after it has been updated at some places. in above example: the sum of A and B is changed by the execution of the transaction if inconsistency occur. Suppose that, just before

the execution of transaction T_i, the values of accounts A and B are \$1000 and \$2000, respectively. Now suppose that, during the execution of transaction T_i, a failure occurs that prevents T_i from completing its execution successfully. Further, suppose that the failure happened after the write(A) and write(B) operation. In this case, the values of accounts A and B are changed. e.g. sum of balances of all accounts, minus sum of loan amounts must unequal value of cash-in-hand.

Consistency Requirement in Above Example : The sum of A and B is unchanged by the execution of the transaction. In general, consistency requirements include. Explicitly specified integrity constraints such as primary keys and foreign keys. Implicit integrity constraints - e.g. sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand. A transaction must see a consistent database. During transaction execution the database may be temporarily inconsistent. When the transaction completes successfully the database must be consistent.

Q.32 ***A university registrar office maintain data about course, course offering, student, instructor, enrolment and grade. Construct an E–R diagram for office and convert E–R diagram into tables.*** ***[R.T.U. 2014]***

Ans. Given that the main entity sets are student, course, course-offering, and instructor. The entity set course-offering is a weak entity set dependent on course. The assumptions made are :

(a) A class meets only at one particular place and time. This E-R diagram cannot model a class meeting at different places at different times.

(b) There is no guarantee that the database does not have two classes meeting at the same place and time.

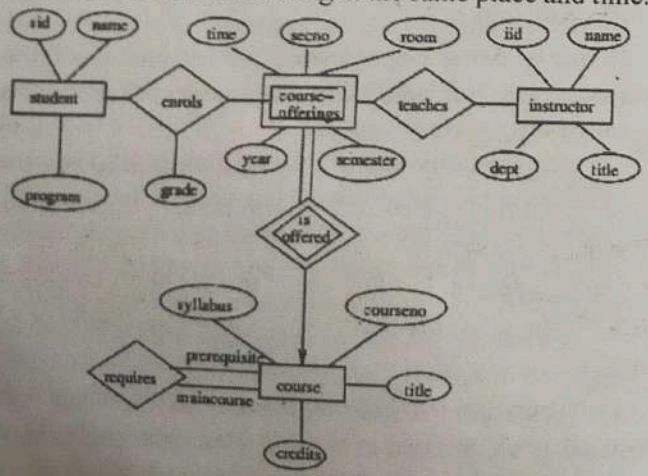


Fig. : E-R diagram for a university

University registrar's tables:

student (student-id, name, program)

course (course-no, title, syllabus, credits)

course-offering (course-no, sec-no, year, semester, time, room)

instructor (instructor-id, name, dept, title)

enrols (student-id, course-no, sec-no, semester, year, grade)

teaches (course-no, sec-no, semester, year, instructor-id)

requires (main-course, pre-requisite)

Table 1 : Student

Student-id	Name	Program
S-112	A	P-1
S-117	B	P-2
S-120	C	P-3

Table 2 : Course

Course-No	Title	Syllabus	Credits
11234	C++	RTU	5
11671	TEF	RTU	7
11841	OOP's	RTU	8

Table 3 : Course-offering

Course-No	Sec-No	Year	Semester	Time	Room
11234	D	First	Fifth	12:30	12
11671	E	Second	Sixth	2:15	13
11841	F	Third	Seventh	3:30	14

Table 4 : Instructor

Instructor-Id	Name	Dept	Title
I-45	AAA	EE	C++
I-86	BBB	EC	TEF
I-13	CCC	CS	OOP's

Table 5 : Enrols

Student-Id	Course-No	Sec-No	Semester	Year	Grade
S-112	11234	D	Fifth	First	a
S-117	11671	E	Sixth	Second	b
S-120	11841	F	Seventh	Third	c

Table 6 : Teaches

Course-No	Sec-No	Semester	Year	Instructor-Id
11234	D	Fifth	First	I-45
11671	E	Sixth	Second	I-86
11841	F	Seventh	Third	I-13

Table 7 : Requires

Main-Course	Pre-Requisite
BCA	12^{th}
MCA	BCA
B.Tech.	12^{th} + Science

Q.33(a) ***Describe the structure of a DBMS. How it is different from RDBMS?*** *[R.T.U. Dec.2013]*

OR

Describe the five components of DBMS environment and discuss how they relate to each other. *[R.T.U. 2012]*

(b) ***Explain DBA and data dictionary in brief.*** *[R.T.U. Dec.2013]*

Ans.(a) Structure of DBMS : *Refer to Q.27.*

Difference between DBMS and RDBMS

S. No.	DBMS	RDBMS
1.	Database Management systems were introduced in 1960s by Charles.	Relational Database Management Systems were in 1970 by Dr. E.F.Codd.
2.	During introduction it followed the navigational modes (Navigational DBMS) for data storage and fetching.	This model uses relationship between tables using primary keys, foreign keys and indexes.
3.	Data fetching is slower for complex and large amount of data.	Comparatively faster because of its relational model.
4.	Used for applications using small amount of data.	Used for complex and large amount of data.
5.	DBMS supports 3 rules of E.F. CODD out off 12 rules.	RDBMS supports minimum 6 rules of E.F. CODD.
6.	DBMS does not support distributed databases.	RDBMS supports distributed databases.
7.	DBMS contains only flat data.	RDBMS contains some relation between entities.
8.	DBMS supports single user.	RDBMS supports multiple user
9.	In DBMS Normalization process will not be present so data redundancy is common in this model.	In RDBMS, normalization process will be present to check the database table consistency. Keys and indexes are used in the tables to avoid redundancy.
10.	Example systems are dBase, Microsoft Access.	Example systems are SQL Server, Oracle.

Ans. (b) Data Base Administrator (DBA) : *Refer to Q.30(b).*

Data Dictionary : Data dictionary stores metadata (data about data) about the structure of the database, in particular the scheme of the database.

A relational-database system needs to maintain data about the relation, such as the schemes of the relation. This information is called the data dictionary or system catalog. Among the types of information that the system must store are these :

(i) Names of the relations
(ii) Names of the attributes of each relation.
(iii) Domains and lengths of attributes
(iv) Names of views defined on the database and definiation of those views.
(v) Integrity constraints.
(vi) Names of authorized users.
(vii) Accounting information about users.
(viii) Passwords of users.
(ix) No. of tuples in each relation.
(x) Method of storage organization.

Q.34 ***What is RDBMS? Describe integrity constraints in relational data model.*** *[R.T.U. 2013]*

Ans. Relational Data Base Management System (RDBMS) : A relational database consists of a collection of tables, each having a unique name. A row in a table represents a relationship among a set of value. A table is a entity set and row is an entity. Thus a table represents a collection of relationships.

There is a direct correspondence between the concept of a table and the mathematical concept of a relation, from which the relational data model takes its name.

Basic Structure

Consider the employee table. It has 3 column headers : emp_ID, emp_Name and salary. If we follows the terminology of the relational data model, we refer to these headers as attributes. For each attributes, there is set of permitted values called the domain of the attribute. For the attribute emp_Name, the domain is the set of all employee names. Let D_1 denote the set of all employee IDs, D_2 the set of all employee name and D_3 the set of all salary.

Then, any row of employee must consist of a 3-tuple (V_1, V_2, V_3) where V_1 is an employee id (*i.e.* V_1 is in domain D_1) V_2 is a employee name (*i.e.* V_2 is in domain D_2) and V_3 is a salary (*i.e.* V_3 is in domain D_3)

$V_1 \in D_1, V_2 \in D_2, V_3 \in D_3$

In general employee will contain only a subset of the set of all possible rows. Therefore, employee is a subset of

$D_1 * D_2 * D_3$

In general a table of n- attributes must be a subset of

$D_1 * D_2 * .... * D_{n-1} * D_n$

$X_{i=1}^{n} D_i$ (all possible rows)

Integrity Constraints

An Integrity constraint is a condition that is enforced automatically by the DBMS and whose violation prevents the data from being stored in the database. The DBMS enforces integrity constraints in that it only permits legal instances to be stored in the database. The key constraints and the referential integrity constraints are identified as the two minimum constraints that must be enforced by the DBMS.

Constraints can be Defined in Two Ways

1. The constraints can be specified immediately after the column definition. This is called column-level definition.
2. The constraints can be specified after all the columns are defined. This is called table-level definition.

Example of Integrity Constraints

1. No two account can have the same account number.
2. An account number cannot be null.

(i) Primary Key : *Refer to Q.3.*

(ii) Referential Integrity : *Refer to Q.2.*

(iii) SQL Not Null Constraint : This constraint ensures all_rows in the table contain a definite value for the column-which is specified as not null. Which means a null value is not allowed.

Syntax to define a not null constraint :

[CONSTRAINT constraint_name] Not Null

(iv) SQL Unique Key : This constraint ensures that a column or a group of columns in each row have a distinct value.

A columns () can have a null value but the values cannot be duplicated.

Syntax to define a unique key at column level :

[CONSTRAINT constraint_name] UNIQUE

Syntax to define a unique key at table level:

[CONSTRAINT constraint_name] UNIQUE (column_name)

(v) SQL Check Constraint : This constraint defines a business rule on a column. All the rows must satisfy this rule. This constraint can be applied for a single column or a group of columns.

Syntax to define a check constraint :

[CONSTRAINT Constraint_name] CHECK (Condition)

Q.35 ***Discuss the various type of keys that are used in relational model.*** *[R.T.U. Dec.2013]*

Ans. Various types of Keys : A key is a single attribute or a combination of attributes whose values uniquely identify each tuple in that relation.

In other words, no two entities in an entity set are allowed to have exactly the same value for all attributes. A key allows us to identity a set of attributes that are sufficient to distinguish relationship from each other.

There are various types of keys :

(i) Super Key
(ii) Candidate Key
(iii) Primary Key
(iv) Alternate Key
(v) Foreign Key

We can understand these keys with the help of following example.

Table : Student

St_Name	St_Roll No.	Aadhar No	Sem	Email	Street	City	Dept No
Dheer	143	71234569	VII	Dheer@xyz.com	Sitapura	Jaipur	5
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table : Department

Dept No.	Dept Name	HOD	Dept_location
5	CS	M.K.	1st Floor
⋮	⋮	⋮	⋮

(i) Super Key : A Super key is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set.

For example :

The St_Roll No. atribute of the student table is sufficient to distinguish are student from another. Thus, St_Roll No. is a superkey. Similarly, the combination of St_Name and St_Roll No. is a super key. The St_Name is not a super key, because several student might have the same name i.e. In our example (student table) following are Super Keys.

St_Roll No

St_Roll No, St_Name ✓
St_Roll No, City ✓
St_Roll No, Sem ✓
St_Name ✗
St_Name, Sem ✗
St_Name, Email ✓
St_Name, DeptNo ✗
Email ✓
Aadhar No. ✓
Aadhar No, St_Name ✓

Suppose that students from same street having different names then

Street ✗
St_Name ✗
St_Name, Street ✓
St_Name, Street, City ✓
St_Name, Street, Sem ✓

(ii) Candidate Key : Minimal superkeys are called candidate keys.

In our example (student table) following are candidate keys.

St_Roll No ✓
Aadhar No. ✓
Email ✓

And if in our system, students from same street are having different names then,

St_Name, Street ✓

But these super keys are not candidate keys

St_Roll No, St_Name ✗ {St_Roll No. alone is sufficient to distinguish.
St_Name, Street, City ✗ {St_Name, street is sufficient to distinguish
St_Roll No., Email ✗ {St_Roll No. or Email is sufficient to distinguish.

i.e. A attribute or a set of minimum attributes which are sufficient to distinguish is called candidate key and combination of other attributes with these attributes can not be a candidate key but it would be a super key.

(iii) Primary Key : The candidate key which is never change or really change is choosen as a primary key.

In our example (student table), among four candidate keys (St_Roll No, St_Name, Street, Email, Aadhar No.) The Aadhar No. never change, so we would select Aadhar no. as a primary key.

(iv) Alternate Keys : The candidate keys which are not choosen as primary key are called alternate keys.

In our example, remaining candidate keys St_Name, Street, Roll No. Email are alternate keys.

(v) Foreign Key : The attribute or set of attributes which are used to link two tables are called foreign key.

In our example, dept No. is a foreign key which is used to link tables student and department. In department table, dept No. is a primary key but it is not a primary key i-student table.

❑❑❑

RELATIONSHIP ALGEBRA AND CALCULAS 2

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *Explain the role of Triggers in SQL programming.* *[R.T.U. 2019]*

Ans. Triggers : A trigger is a statement that the system executes automatically as a side effect of a modification to the database.

Triggers are special type of stored procedures executed automatically when certain events take place. There are different types of triggers for update, for insert and for delete.

Each trigger is associated with a single database table. Triggers are parameterless procedure that are triggered or fired by the event and not by choice. They cannot have parameters to design a trigger mechanism, we must meet two requirements. Those are-

1. Specify when a trigger is to be executed. This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.
2. Specify the actions to be taken when trigger executes.

Q.2 *What is the meaning of sub-query in terms of SQL?* *[R.T.U. 2019]*

Ans. SQL - Sub Queries

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

Q.3 *Consider the relation schema:*
Works(person_name, company_name, salary)
Lives(person_name, street, city)
Located-in(company_name, city)
Managers(person_name, manager_name)
where manager_name refers to person_name.
Give the relational algebra for the following queries:
(i) List the names of the persons work for the company 'SBC' along with the cities they live in.
(ii) Find the name of the persons who live in the same city and same street as their manager.
(iii) Find the persons whose salaries are more than the salary of everybody who works for the company 'SBC'. *[R.T.U. 2015]*

Ans.

(i) $\Pi_{person\_name, city}$ (Lives $\bowtie$ ($\sigma_{company\_name = "SBC"}$ (Works)))

(ii) Π_{person_name} ((Lives $\bowtie$ Managers) $\bowtie_{(manager\_name = Lives2.person\_name \wedge Lives.street = Lives2.street \wedge Lives.city = Lives2.city)}$ (ρ_{Lives2} (Lives)))

(iii) Π_{person_name}(Works) – ($\Pi_{Works.person\_name}$ (Works $\bowtie_{(Works.salaryd \leq Works2.salary \wedge Works2.company\_name="SBC")}$ ρ_{Works2}(Works)))

Q.4 *Explain aggregate operators.*

Ans. Aggregate Operators : The aggregation operators perform mathematical operations like Average, Aggregate, Count, Max, Min and Sum, on the numeric property of the elements in the collection.

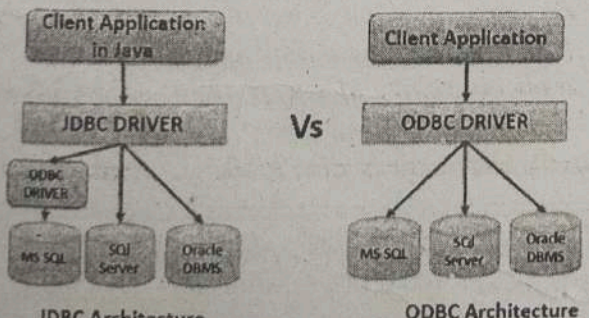
In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

PART-B

Q.5 *What are the difference between JDBC and ODBC?*

[R.T.U. 2019]

Ans. Typically, software applications are written in a specific programming language (such as Java, C#, etc.), while databases accept queries in some other database specific language (such as SQL). Therefore, when a software application needs to access data in a database, an interface that can translate languages to each other (application and database) is required. Otherwise, application programmers need to learn and incorporate database specific languages within their applications. ODBC (Open Database Connectivity) and JDBC (Java DatabBase Connectivity) are two interfaces that solve this specific problem. ODBC is a platform, language and operating system independent interface that can be used for this purpose. Similarly, JDBC is a data API for the Java programming language. Java programmers can use JDBC-to-ODBC bridge to talk to any ODBC compliant database.



JDBC Architecture ODBC Architecture

What is ODBC?

ODBC is an interface to access database management systems (DBMS). ODBC was developed by SQL Access Group in 1992 at a time there were no standard medium to communicate between a database and an application. It does not depend on a specific programming language or a database system or an operating system. Programmers can use ODBC interface to write applications that can query data from any database, regardless of the environment it is running on or the type of DBMS it uses.

Because ODBC driver acts as a translator between the application and the database, ODBC is able to achieve the language and platform independence. This means that the application is relieved of the burden of knowing the database specific language. Instead it will only know and use the ODBS syntax and the driver will translate the query to the database in a language it can understand. Then, the results are returned in a format that can be understood by the application. ODBC software API can be used with both relational and non relational database systems. Another major advantage of having ODBC as a universal middleware between an application and a database is that every time the database specification changes, the software does not need to be updated. Only an update to the ODBC driver would be sufficient.

What is JDBC?

JDBC is a Data API developed for Java programming language. It was released with JDK 1.1 by Sun Microsystems (Java's initial owners). And its current version is JDBC 4.0 (currently distributed with JAVA SE6). Java.sql and javax.sql packages contain the JDBC classes. It is an interface that helps a client to access a database system, by providing methods to query and update data in the databases. JDBC is more suitable for object oriented databases. We can access any ODBC-compliant database by using the JDBC-to-ODBC bridge.

What is the difference between ODBC and JDBC?

ODBC is an open interface which can be used by any application to communicate with any database system, while JDBC is an interface that can be used by Java applications to access databases. Therefore, unlike JDBC, ODBC is language independent. But by using JDBC-to-ODBC bridge Java applications can also talk to any ODBC compliant database.

Comparison Chart

Basis for Comparison	JDBC	ODBC
Basic	JDBC is language and platform dependent (Java Specific).	ODBC is language and platform independent.
Full form	Java Database Connectivity.	Open Database Connectivity.
Code	Code is easy to understand.	Code is complex.

Q.6 ***What do you mean by Null Values? Explain Dynamic SQL in detail.*** *[R.T.U. 2019]*

Ans. Null Values : In databases a common issue is what value or placeholder do you use to represent a missing values. In SQL, this is solved with null. It is used to signify missing or unknown values. The keyword NULL is used to indicate these values. NULL really isn't a specific value as much as it is an indicator. Don't think of NULL as similar to zero or blank, it isn't the same. Zero (0) and blanks " ", are values.

Dynamic SQL : In embedded SQL, the SQL statements to be executed are known at the time of coding the program statements. This is called as static SQL. However, in some cases, the SQL statements to be executed are not known when the program is written. They might be constructed based on user options or, in fact, be written by the user directly in the form SELECT-FROM-WHERE. Such SQL queries, which are not known prior to their execution, from dynamic SQL.

EXECUTE IMMEDIATE <SQL statement>

The EXECUTE IMMEDIATE portion tells the operational environment that the statement belongs to dynamic SQL category.

For example, consider the following dynamic SQL statements. Here, we store the dynamic SQL INSERT statement into a variable called My_statement and pass that variable at the time of execution of dynamic SQL.

My_statement = 'INSERT INTO Student VALUES (10, 'Cryptography', 'Atul', 'THM', 'Security')'

EXECUTE IMMEDIATE: My_statement

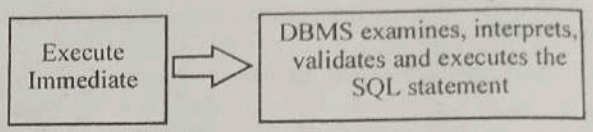
Dynamic SQL is useful in the case of online applications. In such cases, it may not be possible every time to anticipate the kind of queries that the user wants to execute. The user may want to construct them at execution time and execute them immediately. For instance, in a shopping cart application on the Internet, the user may choose to buy 0, 1 or 10 items. In such cases, dynamic SQL would facilitate an elegant creation of a query to process these many items. It may not be possible to achieve the same result using embedded SQL.

There is a variation of the dynamic SQL scheme. We can treat an SQL statement as a temporary object and create it repeatedly. The advantages offered by this approach are as follows :

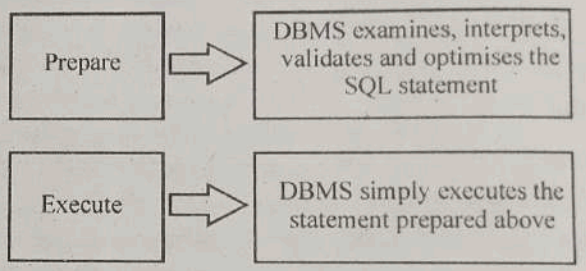
- We can write queries.
- Performance can be far better. This is because the statement is parsed, validated and optimised far ahead of execution time.
- Using parameters, the same statement can be executed in various ways.

In this model, two statements are required : PREPARE and EXECUTE.

The PREPARE statement creates an SQL object that contains the SQL statement to be executed. When this statement is executed, the DBMS examines, interprets, validates and optimises it. It is stored until an EXECUTE (and not EXECUTE IMMEDIATE) statement is encountered. At this stage, the statement is actually executed. This approach is contrasted with the earlier (EXECUTE IMMEDIATE) approach in fig.



(a) Execute Immediate Approach



(b) Prepare-Execute Approach

Fig. : Dynamic SQL approaches

Q.7 ***Write SQL queries for following operations:***

(a) Create student registration table and insert records in it.

(b) Update records based on a key.

(c) Display name and Roll numbers of students who have scored more than 60% marks.

(d) Delete records and table. *[R.T.U. 2017]*

Ans.(a) Create Student registration table and insert records in it :

```
CREATE TABLE Student_Reg
(
 RollNo int,
 Name Varchar(50),
 Course Varchar(10),
 Branch Varchar(10),
```

```
    Score float,
    PRIMARY KEY(RollNo)
    );
    INSERT into Student_Reg
    VALUES (1, Alok, B.Tech, CSE, 70.5);
```

Ans.(b) Update records based on a key

```
UPDATE Student_Reg
SET Course = 'M.Tech'
WHERE Marks > 70.0;
```

Ans.(c) Display name and roll no of students who have scored more than 60% marks

```
SELECT RollNo, Name
FROM Student_Reg
WHERE Marks > 60;
```

Ans.(d) Delete records and table

```
//Delete Records
DELETE from Student_Reg
WHERE Marks < 33
//Delete Table
DROP Student_Reg
```

Q.8 *Consider the employee database given below :*

Employee (emp_name, street, city)
Works (emp_name, company_name, salary)
Company (company_name, city)
Manager (emp_name, manager_name)
Give an expression in SQL for each of the following queries :

(i) Modify the database so that Jones now lives in New town.

(ii) Give all managers of first bank corporation a 10 Percent raise.

(iii) Give all managers of First bank corporation a 10 percent raise unless the salary becomes greater than $100,000. In such cases give only a 3 percent raise.

(iv) Find the names of all employees in the database who live in the same city as the company for which they work. *[R.T.U. 2016]*

Ans.(i)

```
update employee
    set city='newtown'
    where emp_name = 'jones';
```

Ans.(ii)

```
update works T
    set salary = salary + salary * 10/100
    where company-name = 'First Bank Corporation';
```

Ans.(iii)

```
Update works T
    set T.salary = T.salary * 1.03
    where T.emp_name in (salary manager_name
                         from manages)
    and T.salary * 1.1 > 100000
    and T.company_name = First Bank Corporation
    update works T
    set T.salary = T.salary * 1.1
    where T.emp_name in (select manager_name
                         from manages)
    and T.salary * 1.1 <= 100000
    and T.company_name = 'First Bank Corporation'
```

SQL-92 provides a **case** operation, using which a more concise solution can be obtained.

```
update works T
set T.salary = T.salary *
(case
      when (T.salary * 1.1 > 100000) then 1.03
else 1.1
)
where T.emp_name in (select manager_name
from manages) and
T.company_name = 'First Bank Corporation'
```

Ans.(iv)

```
select e.emp_name
    from employee e, works w, company c
    where e.emp_name = w.emp_name and
    c.city = c.city and
    w.company_name = c.company_name
```

Q.9 *Consider the following tables:*

Branch (Branch_No, street, city, pincode)
Staff (Staff_No, Fname, position, sex, DOB, Salary, Branch_No)
Answer the following queries using SQL commands:

(i) List all staff with a salary between Rs. 20,000 and Rs. 30,000 of branch office Delhi or Jaipur.

(ii) Find the number of staff working in each branch and sum of their salaries.

(iii) Find all staff whose salary is larger than the salary of at least one member of staff branch 'BO3'.

(iv) Give all staff a 3% pay increases.

[R.T.U. 2015]

Ans.(i) SELECT * FROM Staff, Branch
WHERE
Staff.Branch_no = Branch.Branch_no
AND Salary BETWEEN 20000 AND 30000
AND City = 'Delhi' OR City='Jaipur'

(ii) SELECT Branch_no, COUNT(Staff_no)
as "staff_count", SUM(Salary)
FROM Staff
GROUP BY Branch_no

(iii) SELECT Staff_no FROM Staff
WHERE Salary > SOME
(SELECT Salary
FROM Staff
WHERE Branch_no='BO3')

(iv) UPDATE Staff
SET Salary = Salary + (0.03 *.Salary)

Q.10 ***How would you use the feature of nested queries in SQL to develop complex queries? Give examples.*** ***[R.T.U. 2015]***

Ans. An SQL subquery is a query that is nested inside another query such as SELECT, INSERT, UPDATE or DELETE. In addition, an SQL subquery can be nested inside another subquery.

An SQL subquery is also called an inner query while the query that contains the subquery is called an outer query.

Let's take a look at the following subquery that returns employees who are located in the offices in the USA.

- The subquery returns all *offices codes* of the offices that locate in the USA.
- The outer query selects the last name and first name of employees whose office code is in the result set returned by the subquery.

Outer Query **Subquery or Inner Query**

```
SELECT lastname, firstname
FROM emplyees
WHERE OfficeCode IN (SELECT officeCode
                     FROM offices
                     WHERE country = 'USA')
```

Another example, we can use comparison operators, like, =, >, <, etc., to compare a single value returned by the subquery with the expression in the WHERE clause.

Consider following payments table :

- Payments(customerNumber, checkNumber, paymentDate, amount)

Consider the following query :

```
SELECT customerNumber,
       checkNumber,
       amount
FROM payments
WHERE amount = (
      SELECT MAX(amount)
      FROM payments
);
```

Q.11 ***Explain Relationship algebra joints.*** ***[R.T.U. 2014]***

[Note : This should be relational algebra joins.]

OR

Discuss the difference between five join operation: theta join, equi join, natural join, outer join and semi join. ***[R.T.U. Dec.2013]***

Ans. Relational Algebra Join Operations

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

Following are the different types of joins:

1. Theta Join
2. Equi Join
3. Natural Join
4. Outer Join
5. Semi Join

A theta join allows for arbitrary comparison relationships (Such as $\geq$).

- An equi join is a theta join using the equality operator.
- A natural join is an equi join on attributes that have the same name in each relationship.
- We will now discuss them one by one to understand the difference between them.

1. Theta (θ) Join

Theta join is the join condition. Theta joins combines tuples from different relations provided they satisfy the theta condition.

Notation :

$R1 \bowtie_\theta R2$

R1 and R2 are relations with their attributes (A1, A2, ..., An) and (B1, B2, ..., Bn) such that no attribute matches that is $R1 \cap R2 = \phi$. Here θ is condition in form of set of conditions C.

Theta join can use all kinds of comparison operators.

Table : Student Relation

SID	Name	Std
101	Dheer Singh	10
102	Saveen	11

Table : Subjects Relation

Class	Subject
10	Math
10	English
11	Music
11	Sports

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

Table : Output of Theta Join

SID	Name	Std	Class	Subject
101	Dheer Singh	10	10	Math
101	Dheer Singh	10	10	English
102	Sawra	11	11	Music
102	Sawra	11	11	Sports

2. Equi-Join

When Theta join uses only **equality** comparison operator it is said to be Equi-Join. The above example corresponds to equi-join.

3. Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way Cartesian product does. Instead, Natural Join can only be performed if the there is at least one common attribute exists between relation. Those attributes must have same name and domain.

Natural join acts on those matching attributes where the values of attributes in both relation is same.

Table 1 : Relation Courses

SID	Course	Dept
CS01	Database	CS
ME01	Mechanics	ME_2
EE01	Electronics	EE

Table 2 : Relation HOD

Dept	Head
CS	Shailesh Aravatiya
ME	Kavita
EE	P. Mangal

Table 3 : Relation Courses ⋈ HOD

Dept	CID	Course	Head
CS	CS01	Database	Shailesh Aravatiya
ME	ME01	Mechanics	Kavita
EE	EE01	Electronics	P.Mangal

4. Outer Joins

All joins mentioned above, that is Theta Join, Equi Join and Natural Join are called inner-joins. An inner-join process includes only tuples with matching attributes, rest are discarded in resulting relation. There exists methods by which all tuples of any relation are included in the resulting relation.

There are three kinds of outer joins :

(a) Left Outer Join (R ⟕ S)

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

Table : Left Relation

A	B
100	Database
101	Mechanics
102	Electronics

Table : Right Relation

A	B
100	Shailesh Aravatiya
102	Kavita
104	P. Mangal

Table : Left Outer Join Output Courses ⟕ HOD

A	B	C	D
100	Database	100	Shailesh Aravatiya
101	Mechanics	—	—
102	Electronics	102	Kavita

(b) Right Outer Join : (R⟖S)

All tuples of the Right relation, S, are included in the resulting relation and if there exists tuples in S without any matching tuples in R then the R-attributes of resulting relation are made NULL.

Table : Right Outer Join Output

A	B	C	D
100	Database	100	Shailesh Aravatiya
102	Electronics	102	Kavita
—	—	104	P. Mangal

(c) Full Outer Join : (R⟗S)

All tuples of the both participating relations are included in the resulting relation and if there no mathcing tuples for both relations, their respective unmatched attributes are made NULL.

Table : Full Outer Join Output Courses ⟗ HOD

A	B	C	D
100	Database	100	Shailesh Arayatiya
101	Mechanics	——	—
102	Electronics	102	Kavita
—	—	104	P. Mangal

5. Semi Join

In semi join, first we take the natural join of two relations then we project the attributes of first table only. So after join and matching the common attribute of both relations only attributes of first relation are projected.

Example :

Table : Faculty

facId	facName	Dept	salary	rank
F234	Monika	CSE	21000	lecturer
F235	Amidya	ENG	23000	Asso Prof
F236	Neelam	CSE	27000	Asso Prof
F237	Manju	IT	32000	Professor

Table : Course

Crs Code	CrS Title	Fld
C3456	TOC	F234
C3457	FM	
C3458	DBMS	F236
C3459	OS	F237

If we take the semi join of two relations faculty and course then the resulting relation would be as under :

Table : Semi join operation on Faculty ⋉ Course

facId	facName	Dept	Salary	Rank
F234	Monika	CSE	21000	lecturer
F236	Neelam	CSE	27000	Asso Prof
F237	Manju	IT	32000	Professor

Q.12 *Consider following schemas :*
Project (Pid, Pname, dept–no)
Works–on (emp–id, Pid, hours)
Employee (emp–id, ename, address, salary)
Department (dept–no, dname)
Write relational algebra syntax for following:

(i) For each employee working on a project with Pname of '231 project', retrieve the name of the employee and his/her salary.

(ii) Retrieve the name of each employee who works on all project controlled by department number 5.

(iii) For each project on which more than two employee work retrieve the project number, the project name and the number of employee who work on the project. *[R.T.U. 2014]*

Ans. (i) $PROJ\_NAME \leftarrow \sigma_{Pname = '231\ project'}(Project)$

$PROJ\_WORKS \leftarrow (PROJ\_NAME \bowtie_{pid = pid} WORKS\text{-}on)$

$WORKS\_EMP \leftarrow (PROJ\_WORKS \bowtie_{emp\_id = emp\_id} Employee)$

$RESULT\ \pi_{ename,\ salary}(WORKS\_EMP)$

(ii) $DEPT5\_PROJS(pid) \leftarrow \pi_{pid}(\sigma_{dept\text{-}no\ =\ 5}(Project))$

$EMP\_PROJ\ (emp\_id, pid) \leftarrow \pi_{emp\_id,\ pid}(WORKS\text{-}on)$

$RESULT\_EMP\_IDS \leftarrow (EMP\_PROJ + DEPT5\_PROJS$

$RESULT \leftarrow \pi_{ename}(RESULT\_EMP\_IDS \times EMPLOYEE)$

(iii) $T_1(pid, No\text{-}of\text{-}employees) \leftarrow {}_{pid}\Im_{COUNT,\ emp\_id}(WORKS\text{-}on)$

$T_2 \leftarrow \sigma_{No\text{-}of\text{-}employees > 2}(T_1)$

$RESULT \leftarrow \pi_{Pname,\ pid}(T_2 \times Project)$

Q.13 *Explain following operations in relational algebra with suitable examples :*
(i) Division
(ii) Grouping

Ans. (i) The Division Operation : The division operation is very useful for some queries that include phrases like "for or "every". Division operation is denoted by ÷ or / and general form of division operation is

$$R \div S \text{ or } R/S$$

Here R and S are two relations and schema of S, S_s is subset of schema of R, R, $S_s \subset R_s$. The result of R ÷ S is a relation defined on the schema $R_s - S_s$, that is it

contains only those attributes of first relation R that are in second relation.

To understand division operation consider two relation instances A and B in which (exactly) two fields x and y and B has just one field y with the same domain as Result of A ÷ B will contain all the x values such that for every y value in B, the a tuple <x, y> in A. Division operation is illustrated in figure.

It helps to think of A as a relation listing emp-id and dept-no where the employee ever worked and of the B relation as listing of departments. Now consider a query employee-id who have worked in all the departments". It is shown as query 19 – result of this shows that emp-id 1976 is the only one who has worked in – three departments or we can say 1976 is the only emp-id value for which we have in A relation with all the deptid values in B relation.

It can be written as

EmpID	DeptNo.
1976	1
1976	4
1976	3
1283	4
1283	3
2211	3

Relation A instance

DeptNo
1
3
4

Relation B instance

EmpID
1976

A/B

(ii) Grouping : Queries may require grouping the tuples in the database on the basis of values stored in some column and then applying aggregate function on groups. Such queries can be solved by using grouping operator denoted by G and then applying the aggregate function. The general form of such queries is

<grouping attributes> $G_{<\text{function list}>}$ (R)

For example, consider a query "find out maximum salary in each department." To solve this query first we have to make groups of employees, DeptNo wise and then find out the maximum salary for each department.

DeptNo $g_{\text{Max (salary)}}$ (Employee)

Consider another example of such queries "Find out the number of employees working in each department and maximum salary for each department". We would write the expression.

DeptNo $g_{\text{count (empid-No), max (salary)}}$ (Employee)

Table 1

DeptNo	Avg of Salary	Dept No
1	15000	1
3	9833	3
4	9667	4

Table 2

Count of Employees	Max of Salary
1	1200
3	15000
3	11500

PART-C

Q.14 ***What is embedded SQL?***

Write the following queries in SQL by considering the employee data base ...

(a) Find all the employees in database who live in the same cities as the companies for which they work.

(b) Find all the employees who earn more than the average salary. *[R.T.U. 2019]*

Ans. Embedded SQL : Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as a host language. Because the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor. Embedded SQL is a robust and convenient method of combining the computing power of a programming language with SQL's specialized data management and manipulation capabilities. The SQL standard defines embedding of SQL as embedded SQL and the language in which SQL queries are embedded is referred as host language .The result of the query is made available to the program one tuple (record) at a time .To identify embedded SQL requests to the preprocessor, we use EXEC SQL statement.

EXEC SQL, embedded SQL statement, END-EXEC

Note : A semi-colon is used instead of END-EXEC when SQL is embedded in C or Pascal.

Embedded SQL statements: declare cursor, open, and fetch statements.

```
EXEC SQL
        declare c cursor for
        select cname, ccity
        from deposit, customer
        where deposit.cname = customer.cname
        and deposit.balance > :amount
        END-EXEC
```

where amount is a host-language variable.

EXEC SQL open c END-EXEC

This statement causes the DB system to execute the query and to save the results within a temporary relation. A series of fetch statement are executed to make tuples of the results available to the program.

Need to Access a Database Using a General Purpose Programming Language : Impedance mismatch is the term used to refer to the problems that occur because of differences between the database model and the programming language model.

Impedance mismatch is less of a problem when a special database programming language is designed that uses the same data model and data types as the database model. One example of such a language is Oracle's PL/SQL. For object databases, the object data model is quite similar to the data model of the Java programming language, so the impedance mismatch is greatly reduced when Java is used as the host language for accessing a Java-compatible object database. Several database programming languages have been implemented as research prototypes.

Most database access in practical situations is through software programs that implement database applications. This software is usually developed in a general purpose programming language such as Java, COBAL or C/C++. The database language such as SQL is, therefore, embedded into general-purpose programming language for accessing the database. When database statements are included in a program, the general-purpose programming language is called the host language, whereas the database language-SRL, in our case is called the data sub-language.

Embedding database commands in a general-purpose programming language : In this approach, database statements are embedded into the host programming language, but they are identified by a special prefix. For example, the prefix for embedded SQL is the string EXEC SQL, which precedes all SQL commands in a host language program. A precompiler or preproccessor scans the source program code to identify database statements and extract them for processing by the DBMS. They are replaced in the program by function calls to the DBMS-generated code. This technique is generally referred to as embedded SQL.

(a) selectE.person_name

fromEmployee as E, Works as W, Company as C

whereE.person_name=W.person_name and E.city=C.city

andW.company_name=C.company_name

(b)SELECT * FROM employees

WHERE salary >

ALL(SELECT avg(salary)FROM employees GROUP BY department_id);

Q.15 ***Explain following operations in relational algebra:***

(a) Selection

(b) Projection

(c) Join

(d) Rename. *[R.T.U. 2017]*

Ans.(a) Select Operation : The select operation selects tuples that satisfies a given predicate. We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parenthesis after the σ.

Loan-number	Branch-Name	Amount
L - 11	Round Hill	900
L - 14	Down town	1500
L -15	Perryridge	1500
L - 16	Perryridge	1300
L - 17	Downtown	1000
L - 23	Red wood	2000
L - 93	Mianus	500

To select those tuples of the loan relation where branch is "perryridge".

$\sigma_{branch\_name} = \text{"Perryridge"}$ (loan)

We can find all tuples in which the amount loan is more than $1200.

$\sigma_{amount} > 1200$ (loan)

Thus to find those tuples pertaining to loan of more than $1200 made by the Perryridge branch,

$\sigma_{branch\_name} = \text{"Perryridge"} <amount> 1200$ (loan)

Ans.(b) Project (π) : We have a schema named the loan scheme before which we give a formal definition of the tuple relational calculus, we return to some of the queries for which we wrote relational.

Table 1 : The loan relation

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1500
L-23	Redwood	2000
L-93	Mianus	500

The Project Operation : Suppose we want to list all loan numbers and the amount of the loans, but do not care about the branch name. The project operation allows us to produce this relation. The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the uppercase Greek letter pi (Π). We list those attributes that we wish to appear in the result as a subscript to Π. The argument relation follows in parenthesis. Thus, we write the query to list all loan numbers and the amount of the loan as :

$$\Pi_{\text{loan-number, amount}} (\text{Loan})$$

Table 2 : Loan number and the amount of the loan

loan-number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1500
L-23	2000
L-93	500

Ans.(c) Natural Join ($\bowtie$) : *Refer to Q. 11.*

Ans.(d) The Rename Operation : The resultant relation obtained from any relation algebra expression does not have any name, that we can use to refer to it. It is sometimes very useful to be able to give name to the result of a relational algebra expression. The rename operator, denoted by the lower case Greek letter rho (ρ) is used to do this task. The general form of rename operation is:

$$\rho_x(E)$$

Here x is the new name given to the resultant relation of relational algebra expression E. Rename operator can be used to get a relation R under a new name as relation name R itself is considered to be a relational algebra expression. For example if we use

$$\rho_{Emp} (\text{Employee})$$

Now we can refer Employee relation with it's new name Emp.

A second form of the rename operations is as follows. Assume that a relational algebra expression E has 'n' columns. Then the expression

$$\rho_x (A_1, A_2, \ldots\ldots A_n) (E)$$

returns the result of expression E under the name x and with the columns renamed to $A_1, A_2 \ldots\ldots A_n$. To illustrate rename operation consider the query 7 "Find names of all the employees working in dept-no 4" we can rewrite this query in two steps as:

$$\rho_{\text{emp-d4}} (\sigma_{\text{Dept-No} = 4} (\text{Employee}))$$
$$\rho_{\text{EName (emp-d4)}}$$

Q.16 ***Explain the difference between relational algebra and relational calculus.*** ***[R.T.U. 2017]***

OR

State the difference between tuple and domain relational calculus. ***[R.T.U. 2017]***

OR

Differentiate relational algebra and relational calculus. ***[R.T.U. 2016]***

Ans. Difference between relational algebra and relational calculus :

Relational Algebra : A basic expression in the relational algebra consists of either one of the following :

(i) A relation in the database

(ii) A constant relation

A constant relation is written by listing its tuples within { }, for example { (A-101, Downtown, 500) (A-215, Mianus, 700) }.

Relational Calculus :

1. Domain Relational Calculus : A second form of relational calculus, called domain relational calculus, uses domain variables that take on values from an attribute domain, rather than values for an entire tuple. The domain relational calculus however is closely related to the tuple relational calculus.

(i) Formal Definitions : An expression in the domain relational calculus is of the form :

$$\{ < x_1, x_2, \ldots x_n > | p(x_1, x_2 \ldots\ldots x_n) \}$$

Where $x_1, x_2 \ldots\ldots x_n$ represent domain variables. 'p' represent a formula composed of atoms. An atom in the domain relational calculus has one of the following forms:

$< x_1, x_2 \ldots x_n > \in r$, where 'r' is a relation on 'n' attributes and $x_1, x_2 \ldots x_n$ are domain variables or domain constants.

- $x \ominus y$, where x and y are domain variables and $\ominus$ is a comparison operator.
- $x \ominus C$, where x is a domain variable, $\ominus$ is a comparison operator and 'C' is a constant in the domain of the attribute for which 'x' is a domain variable.

The formula are composed by these atoms using following rules :

- An atom is formula.
- If p_1 is a formula, then $\neg p_1$ and (p_1) are also formula.
- If p_1 and p_2 are formula, then $\neg p_1 \vee p_2$, $p_1 \vee p_2$ and $p_1 \Rightarrow p_2$ are also formula. If $p_1(x)$ is formula in x, where x is a domain variable, then $\exists x\, p_1(x)$ and $\forall (p_1(x)$ are also formulae.

Example : Find all employee working in dept-no = 3

$\{<e_n\ n,\ d,\ mn,\ d_f,\ s, d_n> \mid < e_n\ n,\ d,\ mn,\ d_f,\ s,\ d_n> \in$ employee $\wedge\ d_n = 3\}$

This query requires all seven columns of employee relation, so we need to seven domain variables that are e_n, n, d, mn, d_f, s,d_n. The formula specifies that the collection of variables must belong to employee and d_n which is domain variable for dept-no attribute must be 3.

(ii) Safety of Expression : In tuple relational calculus that some queries of type $\{t|\neg(t\in R)\}$ are not safe as they produce infinite relation. We can define safety of expression explicity and for that we are defining domain of a tuple relational formula p, dom(p) as the values that appear in tuples of a relation mentioned in p and all the values referenced by p. For example, dom(t∈Employee $\wedge$ salary > 10000) is the set having 10,000 and all the values appearing in employee.

An expression $\{t|(p(t)\}$ is safe if all values of the result are from dom (p).

So the expression $\{t|\neg(t\in R)\}$ is unsafe as dom($\neg(t\in R)$) is the set of all the values appearing in R. (R is relation specified in p). Similar situation may arise in domain relational calculus. An expression such as :

$\{< a, b, c > \mid \neg(< a, b, c) \in R)\}$ is unsafe, as it allows values that are not in the domain of expression. For the domain relational calculus we define some rules for safety of expression.

$\{<x_1, x_2, \ldots, x_n> \mid P(x_1, x_2, \ldots x_n)\}$

(i) All values that appear in tuples of the expression are values from dom (p).

(ii) For every "These exists" subformula of the form $\exists x\ (p_1(x)$, the subformula is true if and only if there is a value x in dom (p_1) such that $p_1(x)$ is true.

(iii) For every 'for all' subformula of the form $\forall x\ (p_1(x))$, the subformula is true if and only if $p_1(x)$ is true for all values x from dom (p_1).

These rules helps in avoiding test of infinitely many posibilities as we restrict our attention only on the values appeared in dom (p).

All the domain relational calculus queries given in the example are safe.

2. Tuple Relational Calculus : When we write a relational-algebra expression, we provide a sequence of procedures that generate the answer to our query. The tuple relational calculus is a non-procedural query language. It describes the desired information without giving a specific procedure for obtaining that information.

(i) Formal Definition : A query in the tuple relational calculus is expressed as :

$\{t \mid p(t)\}$

that is, it is the set of all tuples t such that predicate 'P' is true for t. We use t[A] to denote the value of tuple on attribute A and we use $t \in r$ to denote that tuple 't' is in relation 'r'.

A tuple-relational-calculus formula is built up out of atoms. An atom has one of the following forms :

$S \in r$, where 'S' is a tuple variable and 'r' is a relation.

S[x] – u[y], where S and u are tuple variables. x is an attribute on which 'S' is defined, y is an attribute on which 'y' is defined.

S[x] ⊖C, where 'S' is a tuple variable, x is an attribute on which 'S' is defined, ⊖ is a comparison operator and C is constant in the domain of attribute x.

The tuple relational calculus is restricted to safe expressions is equivalent in expressive power to the basic relational algebra. Thus for every relational expression using only the basic operation.

We build up formulae from atoms using the following rules:

- An atom is a formula.
- If p_1 is a formula, then so are $\neg p_1$ and (p_1).
- If p_1 and p_2 are formulae then so are $p_1 \vee p_2$, $p_1 \wedge p_2$ and $p_1 \Rightarrow p_2$.
- If $p_1(S)$ is a formula containing a free variable and r is a relation, then $\exists S \in r\ (p_1)(S)$ and $\forall S \in r\ (p_1)(S)$ are also formula.

Note : Expressions like $\{t \mid \neg(t\in r)\}$ are called unsafe as it generates infinite tuples. There are infinite tuples not belonging to r.

(ii) Expressive power of Language : Tuple relational calculus restricted to safe expressions and domain relational calculus also restricted to safe expressions are as powerful as relational algebra. If a query language can express all the queries we can express in relational algebra, it is said to be relationally complete. So domain relational calculus and tuple relational calculus are relationally complete.

A practical query langauage is expected to be relationally complete, in addition commercial query languages typically support features that allow us to express some queries that cannot be expressed in relational algebra.

Q.17 ***Explain Triggers with the help of suitable example.*** *[R.T.U. 2017]*

OR

What is trigger? How do we create triggers on a database? Show some syntax. *[R.T.U. 2016, 2014]*

OR

Write short note on Triggers.

[Raj. Univ. 2007, 06, 05]

Ans. Triggers : *Refer to Q.1.*

Triggers are persistent and are accessible to all database operations. Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

Triggers are useful mechanism for alerting humans or for starting certain tasks automatically when certain conditions are met.

Components of Trigger

Part	Description	Possible Value
Trigger Timing	When the trigger Fires in relation to the triggering event	BEFORE AFTER
Triggering event	Which data manipulation operation on the table or view causes the trigger to fire	INSERT DELETE UPDATE
Triggering Type	How many times the trigger body executes	Statement Row
Trigger Body	What action the trigger performs	Complete PL/SQL block

Syntax of Creating Statement Trigger

The syntax for creating a trigger is:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
    {BEFORE | AFTER | INSTEAD OF }
    {INSERT [OR] | UPDATE [OR] | DELETE}
    [OF col_name]
    ON table_name
    [REFERENCING OLD AS o NEW AS n]
    [FOR EACH ROW]
WHEN (condition)
DECLARE
    Declaration-statements
BEGIN
    Executable- statements
EXCEPTION
    Exception-handling- statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing trigger with the trigger name.
- (BEFORE | AFTER | INSTEAD OF) : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE): This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Creating DDL Event Triggers

```
CREATE or REPLACE trigger CREATE_DB_OBJECT_ AUDIT
    after create on schema
    begin
    call INSERT_AUDIT_RECORD (ora_dict_obj_name);
    end;
    /
```

As shown in this example, you can reference system attributes such as the object name.

To protect the objects within a schema you may want to create a trigger that is executed for each attempted drop table command. That trigger will have to be a BEFORE DROP trigger.

Startup and Shutdown Trigger

```
CREATE or REPLACE trigger PIN_ON_STARTUP
after startup on database
begin
DBMS_SHARED_POOL.KEEP (
"SYS.STANDARD", "P");
end;
/
```

This example shows a simple trigger that will be executed immediately after a database startup. You can modify the list of packages in the trigger body to include those most used by your applications.

Startup and shutdown triggers can access the ora_instance_num, ora_database_name, ora_ login_user, and ora_sysevent attributes.

Q.18 ***Explain Embedded SQL and Dynamic SQL.*** *[R.T.U. 2016]*

Ans. Embedded SQL : *Refer to Q.14.*
Dynamic SQL : *Refer to Q.6.*

Q.19 ***(a) Consider following schemas :***
Passengers (Name, Address, Age)
Reservations (Name, FlightNum, Seat)
Flights (FlightNum, DepartCity, DestinationCity, MinutesLate, DepartureTime, ArrivalTime)

(i) Get the name of passengers who had reservation on a flight that was more than 30 minutes late.

(ii) Get the names of passengers who had reservations on all flights that were more than 60 minutes late.

(iii) Get the names of pairs of passengers, who are of the same age.

(b) Discuss various types of inner join operation. *[R.T.U. 2016]*

Ans.(a)

(i) Temp 1 := $\sigma_{\text{MinutesLate}>30}$ Flights
Temp 2 := Reservations/$_{\text{FlightName}=\text{FlightNum}}$ Temp 1
RESULT := Π_{Name} Temp 2

(ii) Temp1 := $\sigma_{\text{MinutesLate}>60}$ Flights
Temp 2 := $\Pi_{\text{Flight Num}}$ Temp 1
Temp 3 := $\Pi_{\text{Name, FlightNum}}$ Reservations
RESULT := Temp 3/Temp 2

(iii) Temp 0 := Passenger [Name1, Address1, Age1]
Temp 1 := Passenger × Temp0
Temp 2 := $\sigma_{\text{Age}=\text{Age1}}$ AND$_{\text{Name}<>\text{Name1}}$ Temp1
RESULT : = $\Pi_{\text{Name1, Name2}}$ Temp 2

Ans.(b) An inner join requires each row in the two joined tables to have matching column values and is a commonly used join operation in applications but should not be assumed to be the best choice in all situations. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join predicate. When the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of A and B are combined into a result row. There are mainly two types of inner join operations :

Equi Join : An equi-join is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as<) disqualifies a join as an equi-join. We can write equi-join as below.

```
SELECT *
FROM employee, department
WHERE employee.
Department ID = department. DepartmentID;
```

Natural Join : The natural join is a special case of equi-join. Natural jon $(\bowtie)$ is a binary operator that is written as $(R \bowtie S)$ where R and S are relations. The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names. The natural join is arguably one of the most important operators since it is the relational counterpart of logical AND. Example of Natural Join-

```
SELECT*
FROM employee NATURAL JOIN
department;
```

Q.20 ***Discuss the various fundamental operations in relational algebra with suitable example.***

OR

Explain expressive power of algebra and calculus.

[R.T.U. 2015]

[R.T.U. Dec.2013, 2011, 2008, Raj. Univ. 2005, 2003, 2001]

Ans. Expressive power of algebra and calculus :

The expressive power of basic relational algebra is

- **Select (σ) : Unary Operation :** The selection, project and rename operations are called unary operations, because they operate on one relation.

(i) Select Operation : *Refer to Q.15(a).*

(ii) Project (π) : *Refer to Q.15(b).*

(iii) Rename (ρ) : *Refer to Q.15(d).*

- **Binary Operations :** Join and Division are example of binary operation. These operate on two relations.

(i) Division (÷) : The division operation denoted by '÷' is suited to queries that include the phrase "for all"

$r \div s$

Let r and s be relations on schemas R and S respectively where

$$R = (A_1 \ldots\ldots, A_m, B_1 \ldots\ldots, B_n)$$

$$S = (B_1 \ldots\ldots, B_n)$$

The result of $r \div s$ is a relation on schemas

$$R - S = (A_1 \ldots\ldots, A_m)$$

$$r \div s = \{t / t \in \pi_{R-S}(r) \wedge \forall u \in S(tu \in r)\}$$

Ex : Relations r, s:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
∈	6
∈	1
β	2

r

B
1
2

S

$r \div s$ =

A
α
β

(ii) The Natural Join Operation (⋈) : The natural join is a binary operation that allows us to combine certain selection and a cartesian product into one operation. It is denoted by the join symbol '⋈'. The natural join operation forms a cartesian forcing equality and those attributes that appear in both relation schemas and finally removes duplicate attributes.

A frequent type of Join connects two relations by:

(i) Equating attributes of the same name and

(ii) Projecting out one copy of each pair of equated attributes i.e. remove duplicate attributes called natural Join

denoted by $R3 = R1 \bowtie R2$

Ex: Sells

Shop	Candy	Price
Sharma	Eclairs	2.50
Sharma	Swaad	2.75
Gupta	Eclairs	2.80
Gupta	Minto	3.00

Shop

Shop	address
Sharma's	Malviya Nagar
Gupta's	Bajaj Nagar

Shoppings = Sells ⋈ Shops

Shoppings

Shops	Candy	Price	Address
Sharma's	Eclairs	2.50	Malviya Nagar
Sharma's	Swaad	2.75	Malviya Nagar
Gupta's	Eclairs	2.50	Bajaj Nagar
Gupta's	Minto	3.00	Bajaj Nagar

(iii) Union (∪) : Consider a query to find the names of all bank customers who have either an account or a loan or both. Note that the customer relation does not contain the information, since a customer does not need to have either an account or a loan at the bank. To answer this query, we need the information in the depositor relation (Table 3) and in the borrower relation (Table 4). We know how to find the names of all customers with a loan in the bank:

$$\Pi_{\text{customer-name}}(\text{borrower})$$

We also know how to find the names of all customers with an account in the bank:

$$\Pi_{\text{customer-name}}(\text{depositor})$$

To answer the query, we need the **union** of these two sets that is, we need all customer names that appear in either or both of the two relations. We find these data

by the binary operation union, denoted, as in set theory, by ∪. So the expression needed is

$$\Pi_{\text{customer-name}}(\text{borrower}) \cup \Pi_{\text{customer-name}}(\text{depositor})$$

The result relation for this query appears in figure. Notice that there are 10 tuples in the result, even though there are seven distinct borrowers and six depositors. This apparent discrepancy occurs because Smith, Jones and Hayes are borrowers as well as depositors. Since relations are sets, duplicate values are eliminated.

Table : The Depositor Relation

customer-name	account-number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Table : The Borrower Relation

customer-name	loan-number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Williams	L-17

customer-name

Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

Fig. : Names of all customers who have either a loan or an account.

(iv) Set Diff. (–) : The set difference operation is used to find tuples that are in one relation but are not in another.

(v) Cartesian Product (×) : The cartesian product operation, denoted by a cross (×), allows us to combine information from any two relation. We write the cartesian product of relations r_1 and r_2 as $r_1 \times r_2$.

For example

The relation schema for r = borrower × loan is

(borrower. customer_name, borrower.loan_number, loan.branch_name, loan. loan-number, loan.amount)

With this schema, we can distinguish borrow.loan_number from loan. loan_number.

(vi) Set intersection (∩) : $r \cap S = r - (r - S)$

The extended relational algebra allows

(a) Generalized Projection (g) : It extends the projection operation by allowing arithmetic functions to be used in the projection list.

(b) Aggregate Functions : It allows G_{sum}, G_{ang}, G_{min}, G_{max} and $G_{dint\text{-}district}$ functions.

(c) Outer-join : *Refer to Q.11.*

Q.21 ***Consider the schemas and write the SQL syntax for mention statements (i), (ii) and (iii).***

Project (Pid, Pname, dept–no)

Works–on (emp–id, Pid, hours)

Employee (emp–id, ename, address, salary)

Department (dept–no, dname)

(i) For each employee working on a project with Pname of '231 Project', retrieve the name of the employee and his/her salary.

(ii) Retrieve the name of each employee who works on all project controlled by department number 5.

(iii) For each project on which more than two employee work, retrieve the project number, the project name and the number of employee who work on the project.

[R.T.U. 2014]

Ans.(i)

```
SELECT    ename, salary
FROM Employee, Works-on, Project
WHERE emp-id = emp-id AND pid = pid
AND Pname = '231 Project';
```

(ii)

```
SELECT    ename
FROM      Employee
WHERE     ((SELECT pid
          FROM     Works-on
```

```
                WHERE   emp-id = emp-id
                CONTAINS
                (SELECT  pid
                FROM     Project
                WHERE    dept-no = 5));
(iii) SELECT     pid, Pname, COUNT (*)
      FROM       Project, Works-on
      WHERE      pid = pid
      GROUP BY   pid, Pname
      HAVING     COUNT (*) > 2;
```

Q.22 ***What is JDBC? Explain establishing a connection to the database, create statement, execute query and iterate result set in JDBC.***

[R.T.U. 2014]

Ans. JDBC : *Refer to Q.5.*

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database.

(1) Establish a JDBC Connection : The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps :

(i)Import JDBC Packages : The Import statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code. To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following imports to your source code.

```
import java.sql.* ; // for standard JDBC programs
import java.math.* ; // for BigDecimal and BigInteger support
```

(ii)Register JDBC Driver : You must register the driver in your program before you use it. Registering the driver is the process by which the Oracle driver's class file is loaded into the memory, so it can be utilized as an implementation of the JDBC interfaces.

You can register a driver in one of two ways.

Approach I - Class.forName() : The most common approach to register a driver is to use Java's Class.for Name() method, to dynamically load the driver's class file into memory, which automatically registers it.

Approach II - DriverManager.registerDriver() : The second approach you can use to register a driver, is to use the staticDriverManager.registerDriver() method.

(iii)Database URL Formulation : After you've loaded the driver, you can establish a connection using theDriverManager.getConnection() method. For easy reference, let me list the three overloaded DriverManager.getConnection() methods :

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

(iv) Create Connection Object

We have listed down three forms of DriverManager.getConnection() method to create a connection object.Once a connection is obtained we can interact with the database. The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database.

(2) Creating Statement : Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement( ) method, as in the following example :

```
stmt = conn.createStatement( );
```

(i)The PreparedStatement Objects

The PreparedStatement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object.

This statement gives you the flexibility of supplying arguments dynamically.

```
pstmt = conn.prepareStatement(SQL);
```

(ii)The CallableStatement Objects

Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute

a call to a database stored procedure. Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.

```
CallableStatement cstmt = null;
String SQL = "{call getEmpName (?, ?)}";
cstmt = conn.prepareCall (SQL);
```

(3) Execute Query

Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

- **boolean execute (String SQL):** Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate (String SQL):** Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected : for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery (String SQL):** Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

(4) Result Set

The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The java.sql.ResultSet interface represents the result set of a database query.

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface can be broken down into three categories :

- **Navigational Methods :** Used to move the cursor around.
- **Get Methods :** Used to view the data in the columns of the current row being pointed by the cursor.
- **Update Methods :** Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

The cursor is movable based on the properties of the ResultSet. These properties are designated when the corresponding Statement that generates the ResultSet is created.

Q.23 ***(a) Consider the following database schema***

employee (ename, street, city)

Works (ename, company_name, salary)

Company (company_name, city)

manager (ename, mgr_name)

Give regular expression for each of the following statements :

(i) Find the company with most employees.

(ii) Find the company with the smallest payroll.

(iii) Give all employees of ABC company a 10-percent salary raise.

(iv) Delete all tuples in the works relation for employee of XYZ, company.

(v) Find the names and cities of residence of all employees who works for ABC company.

(b) Explain any 3 set-operators and 3-Aggregate operators in SQL with the help of suitable examples. ***[R.T.U. 2013]***

Ans. (a) Regular Expression is :

(i) $t_1 \leftarrow {}_{company\_name}G_{count\text{-}distinct}\ ename\ (works)$

$t_2 \leftarrow \max_{num\_employees} (\rho_{company\_strength\ (company\_name,\ num\_employees)} (t_1))$

$\Pi_{company\_name} (\rho_{t3\ (company\_name,\ num\_employees)} (t_1) \bowtie \rho_{t4(num\_employees)} (t_2))$

(ii) $t_1 \leftarrow {}_{company\_name}G_{Sum}$ salary (works)

$t_2 \leftarrow \min_{payroll} (\rho_{company\_payroll\ (company\_name,\ payroll)} (t_1))$

$\Pi_{company\_name} (\rho_{t3\ (company\_name,\ payroll)} (t_1) \bowtie \rho_{t4\ (payroll)} (t_2))$

(iii) works $\leftarrow \Pi_{ename,\ company\_name,\ 1.1*salary}(\sigma_{(company\_name = "ABC")}$ (works) $\cup$ (works $- \sigma_{company\_name = "ABC"}$ (works))

(iv) works $\leftarrow$ works $- \sigma_{company\_name = "XYZ"}$ (works)

(v) $\Pi_{ename,\ city}$ (employee $\bowtie$ ($\sigma_{company\_name = "ABC"}$ (works))

Ans. **(b) Set Operations**

The SQL operations **union, intersect**, and **expect** operate on relations and correspond to the relational-algebra operations $\cup$, $\cap$, and $-$. Like union, intersection, and set difference in relational algebra, the relations participating in the operations must be compatible; that is, they must have the same set of attributes.

Let us demonstrate how several of the example queries can be written in SQL. We shall now construct queries involving the **union, intersect**, and **except** operations of two sets: the set of all customers who have an account at the bank, which can be derived by

select *customer_name*

from *depositor*

and the set of customers who have a loan at the bank, which can be derived by

select *customer_name*

from *borrower*

In our discussion that follows, we shall refer to the relation obtained as the result of the preceding queries as d and b, respectively.

1. The Union Operation

To find all the bank customers having a loan, an account, or both at the bank, we write

(**select** *customer_name*

from *depositor*)

union

(**select** *customer_name*

from *borrower*)

The **union** operation automatically eliminates duplicates, unlike the **select** clause. Thus, in the preceding query, if a customer-say, jones-has several accounts or loans (or both) at the bank, then Jones will appear only once in the result. If we want to retain all duplicates, we must write **union all** in place of **union** :

(**select** *customer_name*

from *depositor*)

union all

(**select** *customer _name*

from *borrower*)

The number of duplicate tuples in the result is equal to the total number of duplicates that appear in both *d* and *b*. Thus, if Jones has three accounts and two loans at the bank, then there will be five tuples with the name Jones in the result.

2. The intersect Operation

To find all customers who have both a loan and an account at the bank, we write

(**select distinct** *customer_name*

from *depositor*)

intersect

(**select distinct** *customer_name*

from *borrower*)

The **intersect** operation automatically eliminates duplicates. Thus, in the preceding query, if a customer–say, Jones–has several accounts and loans at the bank, then loans will appear only once in the result.

If we want to retain all duplicates, we must write intersect all in place of intersect :

(**select** *customer_name*

from *depositor*)

intersect all

(**select** *customer_name*

from *borrower*)

The number of duplicate tuples that appear in the result is equal to the minimum number of duplicates in both *d* and *b*. Thus, if Jones has three accounts and two loans at the bank, then there will be two tuples with the name Jones in the result.

3. The Except Operation

To find all customers who have an account but no loan at the bank, we write

(**select distinct** *customer_name*

from *depositor*)

except

(**select** *customer_name*

from *borrower*)

The **except** operation automatically eliminates duplicates. Thus, in the preceding query, a tuple with customer name Jones will appear (exactly once) in the result only if jones has an account at the bank, but has no loan at the bank.

If we want to retain all duplicates, we must write except all in place of except :

(**select** *customer_name*

from *depositor*)

except all

(**select** *customer_name*

from *borrower*)

The number of duplicate copies of a tuple in the result is equal to the number of duplicate copies of the tuple in depositor minus the number of duplicate copies of the tuple in borrower , provided that the difference is positive. Thus, if Jones has three accounts and one loan at the bank, then there will be two tuples with the name Jones in the result. If, instead, this customer has two accounts and three loans at the bank, there will be no tuple with the name Jones in the result.

The aggregate operators supported by SQL are :

COUNT, SUM, AVG, MIN, MAX

(i) COUNT (A): The number of values in the column A

(ii) SUM (A): The sum of all values in column A

(iii) AVG (A): The average of all values in column A

(iv) MAX(A): The maximum value in column A

(v) MIN(A): The minimum value in column A

Using the COUNT operator

Count the number of sailors

```
SELECT COUNT (*)
FROM Sailors S;
```

Count the number of different sailor names

```
SELECT COUNT (DISTINCT S.sname)
FROM Sailors S;
```

Example of SUM operator

Find the Sum of ages of all sailors with a rating of 10

```
SELECT SUM (S. age)
FROM Sailors S
WHERE S. rating = 10;
```

Example of AVG Operator

Find the average age of all sailors with rating 10

```
SELECT AVG (S. age)
FROM Sailors S
WHERE S. rating = 10
```

❑❑❑

SCHEMA REFINEMENT AND NORMAL FORMS 3

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *What is the purpose of normalization in DBMS?*

[R.T.U. 2019]

OR

Discuss the need of normalization.

Ans. Need of Normalization : When you normalize a database, you have four goals: arranging data into logical groupings such that each group describes a small part of the whole; minimizing the amount of duplicate data stored in a database; organizing the data such that, when you modify it, you make the change in only one place; and building a database in which you can access and manipulate the data quickly and efficiently without compromising the integrity of the data in storage.

Data normalization helps you design new databases to meet these goals or to test databases to see whether they meet the goals. Sometimes database designers refer to these goals in terms such as data integrity, referential integrity, or keyed data access. Ideally, you normalize data before you create database tables. However, you can also use these techniques to test an existing database.

Q.2 *Decompose the schema R = (A, B, C, D, E) into (A, B, C) and (A, D, E). Also show that this decomposition is a lossless-join decomposition if the following set F of functional dependencies holds :*

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

[R.T.U. 2015, 2013]

OR

Suppose we decompose the scheme R (A, B, C, D, E,) into (A, B, C) and (A, D, E). Show that this is a lossless-join decomposition if the Set F = {A → BC, CD→E, B→D, E→A} of functional dependencies hold. *[R.T.U. 2011]*

Ans. A decomposition $\{R_1, R_2\}$ is lossless-join decomposition if

$$R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2$$

Let $R_1 = (A, B, C)$ and $R_2 = (A, D, E)$

So, $R_1 \cap R_2 = A$

Since A is a candidate key. There $R_1 \cap R_2 \rightarrow R_1$ so this is a lossless-join decomposition.

Q.3 *What is normalization.*

Ans. Normalization : Database normalization or data normalization, is a technique to organize the contents of the tables for transactional databases and data warehouses. Normalization is part of successful database design; without normalization, database systems can be inaccurate, slow, and inefficient, and they might not produce the data you expect.

Q.4 *Write advantages of normaliztion.*

Ans. Advantages of Normalization:

1. Avoids data modification (INSERT/DELETE/ UPDATE) anomalies as each data item lives in one place.
2. Greater flexibility in getting the expected data.
3. Normalization is conceptually cleaner and easier to maintain and change as your needs change.
4. Fewer null values and less opportunity for inconsistency.
5. A better handle on database security.

Q.5 *Explain Decomposition.*

Ans. Decomposition : A functional decomposition is the process of breaking down the functions of an organization into progressively greater (finer and finer) levels of detail. In decomposition, one function is described in greater detail by a set of other supporting functions.

The decomposition of a relation scheme R consists of replacing the relation schema by two or more relation schemas that each contain a subset of the attributes of R and together include all attributes in R.

Decomposition helps in eliminating some of the problems of bad design such as redundancy, inconsistencies and anomalies.

There are two types of decomposition :

1. Lossy Decomposition
2. Lossless Join Decomposition

PART-B

Q.6 What is Normalization? Also explain functional dependencies with a suitable example.

[R.T.U. 2019]

Ans. Normalization : *Refer to Q.3.*

Functional Dependencies : The single most important concept in a relational schema design is that of functional dependency. A functional dependency is a constraint between two set of attributes in a relation from a database.

Given a relation R a set of attributes x in R is said to functionally determine. Another attribute y also in R, (written $x \rightarrow y$) if and only if each x value is associated with at most one y value. We call x the determinant set and y the dependent attribute. Thus given a tuple and the values of the attribute in x. One can determine the corresponding value of the y attribute.

A functional depending set 'S' is irreducible if the set has three following properties :

1. Each right set of a functional dependency of 'S' contains only one attribute.
2. Each left set of functional dependency of 'S'is irreducible. It means that reducing any one attribute from left set would not change the content of S.
3. Reducing any functional dependency will change the content of S.

A functional dependency, denoted by $x \rightarrow y$, between two sets of attributes x and y that are subsets of the attributes of relation R.

Functional dependencies and keys : We say that a set of one or more attributes

{A_1, ... A_n} is a key for a relation R if :

1. Those attributes functionally determine all other attributes of the relation.
2. No proper subset of those attributes functionally determines all other attributes of R.

A set of attributes that contains a key is called a superkey. Thus every key is a superkey but not every key is minimal. If a relation has more than one key, one of the keys is designed as the **primary key**.

Closures : Let a relation 'R' has some functional dependencies 'F' specified. The closure is the set of all functional dependencies that may be logically derived from F.

'F' is the set of most obvious and important functional dependencies and F^+, the closure is the set of all the functional dependencies including F and those that can be deduced from F^+. The closure is important and may be needed in finding one or more candidate keys of the relation.

A set of rules that may be used to infer additional dependencies was proposed by Armstrong in 1974. These rules are complete set of rules called Axioms or inference rules such that all possible functional dependencies may be derived from them. The rules are :

1. **Reflexivity Rule :** If 'x' is a set of attributes and 'y' is a subset of 'x', then $x \rightarrow y$ holds.
2. **Augmentation Rule :** If $x \rightarrow y$ holds and 'ω' is the set of attributes, then $\omega x \rightarrow \omega y$ holds.
3. **Transitivity Rule :** If $x \rightarrow y$ and $y \rightarrow z$ hold, then $x \rightarrow z$ holds.

These rules are called Armstrong's Axioms.

The most important additional axioms are :

1. **Union Rule :** If $x \rightarrow y$ and $x \rightarrow z$ hold then $x \rightarrow yz$ holds.
2. **Decomposition Rule :** If $x \rightarrow yz$ holds, so do $x \rightarrow y$ and $x \rightarrow z$.
3. **Pseudotransitivity Rule :** If $x \rightarrow y$ and $\omega y \rightarrow z$ hold then so does $\omega x \rightarrow z$.

Full Functional Dependencies (FFD):

A functional dependency $X \rightarrow Y$ is full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

If all the non-key attributes of the entity completely and functionally depend on the key attribute of the same entity so it is known as full Functional Dependencies.

For example, we have Student table (Entity) having 4 column (attribute).

(1) Sid
(2) Sname
(3) Add
(4) CourseName

The Sname, Add and CourseName are the non-key attribute which completely depend on Sid (Key Attribute).

If we want to retrive any information about student we only need key attribute i.e. Sid and rest of the information we can get on the basis of key attribute.

So here all the attribute (Sname, Add, CourseName) fully depend on key attribute, and hence it is called full Fuctional Dependencies.

Q.7 ***Explain functional dependencies with the help of suitable examples.*** ***[R.T.U. 2017]***

OR

Describe the concept of full functional dependency. ***[R.T.U. 2015]***

OR

Define Functional Dependency. Explain Armstrong's axioms or rules, with examples. ***[R.T.U. 2015]***

OR

Describe the concept of full functional dependency (FFD). ***[R.T.U. Dec. 2013, 2008]***

Ans. Functional Dependencies : *Refer to Q.6.*

Q.8 ***What is Decomposition? Explain Lossy and Lossless join decomposition.*** ***[R.T.U. 2016]***

Ans. Decomposition : *Refer to Q.5.*

Lossy Decomposition : "The decomposition of relation R into R1 and R2 is **lossy** when the join of R1 and R2 does not yield the same relation as in R."

One of the disadvantages of decomposition into two or more relational schemes (or tables) is that some information is lost during retrieval of original relation or table.

Consider that we have table STUDENT with three attributes roll_no , sname and department.

STUDENT:

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relations no_name and name_dept :

No_name **Name_dept**

Roll_no	Sname
111	parimal
222	parimal

Sname	Dept
parimal	COMPUTER
parimal	ELECTRICAL

In lossy decomposition ,spurious tuples are generated when a natural join is applied to the relations in the decomposition.

stu_joined :

Roll_no	Sname	Dept
111	parimal	COMPUTER
111	parimal	ELECTRICAL
222	parimal	COMPUTER
222	parimal	ELECTRICAL

The above decomposition is a bad decomposition or Lossy decomposition.

Lossless Join Decomposition : "The decomposition of relation R into R1 and R2 is **lossless** when the join of R1 and R2 yield the same relation as in R."

A relational table is decomposed (or factored) into two or more smaller tables, in such a way that the designer can capture the precise content of the original table by joining the decomposed parts. This is called lossless-join (or non-additive join) decomposition.

This is also refferd as non-additive decomposition.

The lossless-join decomposition is always defined with respect to a specific set F of dependencies.

Consider that we have table STUDENT with three attribute roll_no , sname and department.

STUDENT :

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

This relation is decomposed into two relation Stu_name and Stu_dept :

Stu_name **Stu_dept**

Roll_no	Sname
111	parimal
222	parimal

Roll_no	Dept
111	COMPUTER
222	ELECTRICAL

Now ,when these two relations are joined on the common column 'roll_no' ,the resultant relation will look like stu_joined.

stu_joined :

Roll_no	Sname	Dept
111	parimal	COMPUTER
222	parimal	ELECTRICAL

In lossless decomposition, no any spurious tuples are generated when a natural joined is applied to the relations in the decomposition.

Q.9 ***We are given a schema S = (A, B, C, D, E). The F of functional dependencies is {A→B, BC→E, ED→A}***

(a) List all candidate keys for S.

(b) Is S in 3NF?Why?

(c) Is S in BCNF? Why?

(d) Find canonical cover F_c of F.

[R.T.U. 2016, 2014]

Ans. (a) Diagrammatic representation of the FD's is

$A \rightarrow B$

$C \rightarrow E$

$D \rightarrow A$

Finding the candidate keys (CKs)

'A' uniquely can't determine all the attributes. It needs 'C' and 'D'. Therefore CK is {A, C, D}.

Similarly if we have 'B' and 'C', we need 'D' to determine all other attributes. So CK is {B, C, D}.

Finally, if we have 'E' and 'D', we need 'C' to determine all other attributes.

So, CK is {C, E, D}

Thus, we have 3CKs

{A, C, D}

{B, C, D}

{C, E, D}

Non-prime attributes are 'C' and 'D'.

(b) Now to check all NFs

(i) Any given relation by definition of a relation is already in 1NF.

(ii) To find out 2NF

Is there any composite key?

Ans. Yes

Then does any part of our composite keys independently determine any of the non-prime attributes?

Ans. 'C' and 'D' cannot be independently determined by any part of any CK. So 2NF condition is passed.

(iii) To find out 3NF

Does the relation have any non-primes?

Ans. Yes

Then, is there a non-prime that determines (point to) another non-prime?

Ans. Neither 'C' determines (points to) 'D' nor vice versa.

So, 3NF condition is passed.

(c) To find out BCNF

In all given FDs, are all the determinents also candidate keys?

No, neither 'A' nor 'BC' nor 'ED' are candidate keys. Therefore BCNF condition does not pass.

$\therefore$ Highest NF is 3NF

(d) In the given dependencies no any redundency. So the canonical cover F_c of F is

$A \rightarrow B$

$C \rightarrow E$

$D \rightarrow A$

Q.10 ***Compute F^+ of the following set F of Functional Dependency for relation schema R = (A,B,C,D,E)***

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

List the candidate key for R. *[R.T.U. 2013]*

Ans. Note : It is not reasonable to expect students to enumerate all of F^+. Some shorthand representation of the result should be acceptable as long as the nontrivial members of F^+ are found.

Starting with $A \rightarrow BC$, we can conclude: $A \rightarrow B$ and $A \rightarrow C$

Since $A \rightarrow B$ and $B \rightarrow D$, $A \rightarrow D$	(decomposition, transitive)
Since $A \rightarrow CD$ and $CD \rightarrow E$, $A \rightarrow E$	(union, decomposition, transitive)
Since $A \rightarrow A$, We have	(reflexive)
$A \rightarrow ABCDE$ from the above steps	(union)
Since $E \rightarrow A$, $E \rightarrow ABCDE$	(transitive)
Since $CD \rightarrow E$, $CD \rightarrow ABCDE$	(transitive)
Since $B \rightarrow D$ and $BC \rightarrow CD$, $BC \rightarrow ABCD$	(augmentative, transitive)

Also, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow D$, etc.

Therefore, any functional dependency with A, E, BC, or CD on the left hand side of the arrow is in F^+, no

matter which other attributes appear in the FD. Allow * to represent any set of attributes in R, then F^+ is $BD \rightarrow B$, $BD \rightarrow D$, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow BD$, $B \rightarrow D$, $B \rightarrow B$, $B \rightarrow BD$, and all FDs of the form $A^* \rightarrow \alpha$, $BC^* \rightarrow \alpha$, $CD^* \rightarrow \alpha$, $E^* \rightarrow \alpha$ where α is any subset of {A, B, C, D, E}. The candidate keys are A, BC, CD, and E.

PART-C

Q.11 ***Explain Boyce-Codd normal form and 3-NF in detail.*** *[R.T.U. 2019]*

Ans.BCNF (Boyce-Code Normal Form)

One of the most desirable normal forms that we can obtain is Boyce-Code Normal Form (BCNF). A relation schema R is in BCNF with respect to a set of functional dependencies if for all functional dependencies in F^+ of a form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds :

- $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$).
- α is a superkey for schema R.

A database design is in BCNF if each member of the set of relation schema that constitutes the design is in BCNF.

Often testing of a relation to see if it satisfies BCNF can be simplified.

- To check if a nontrivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, compute $\alpha+$ and verify that it includes all attributes of R; that is, it is a superkey of R.
- To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than to check all dependencies in F^+.

Example of BCNF

Stud-ID	S-Name	Subject	Grade
1001	Axay	Physics	A
1001	Axay	Chemistry	C
1001	Axay	Maths	C
1002	Sparsh	Physics	A
1002	Sparsh	Chemistry	A
1002	Sparsh	Maths	B

In this relation following FD_s exist :

S_Name, Subject → Grade

Stud_ID, Subject → Grade

S_Name → Stud_ID

Stud_ID → S_Name

Moreover, in this relation two candidate keys (S_Name, Subject) and (Stud_ID, Subject) exist, which are composite keys and contain a common attribute subject. This relation is in 3NF, however a lot of data repetition is there in terms of S-Name and Stud_ID. So for this relation to be in BCNF, we will have to do the decomposition. The rule for decomposition for a relation R, which is not in BCNF, is as follows :

Let $x \subseteq R$, A be the single attribute in R_1 and $x \rightarrow A$ be a FD that causes a violation of BCNF. Decompose R into R – A and X_A.

So for above relation to be in BCNF, the decomposition will be as follows :

Stud_ID	S_Name
1001	Axay
1002	Sparsh

Third Normal Form (3NF) : There are schemas where a BCNF decomposition cannot be dependency preserving. For such schemas, we have two alternatives if we wish to check if an update violates any functional dependencies :

- Pay the extra cost of computing joins to test for violations.
- Use an alternative decomposition, third normal form (3NF), which we present below, which makes testing of updates cheaper, unlike BCNF, 3NF decompositions may contain some redundancy in the decomposed schema.

Definition : BCNF requires that all nontrival dependencies be of the form $\alpha \rightarrow \beta$, where α is a superkey. 3NF relates this constraint slightly by allowing nontrivial functional dependencies whose left side is not a superkey.

A relation schema R is in third normal form (3NF) with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$ at least one of the following holds:

- $\alpha \rightarrow \beta$ is a trivial functional dependency.
- α is a separately for R.
- Each attribute A in $\beta \rightarrow \alpha$ is contained in a candidate key for R.

The two alternatives are the same as the two alternatives in the definition of BCNF. The third alternative of the 3NF definition seems rather unintuitive and it is not obvious why it is useful. It represents, in some sense, a

minimal relaxation of the BCNF conditions that helps to ensure that every schema has a dependency preserving decomposition into 3NF. Its purpose will become more clear.

Third Normal Form :

Following synthesis algorithm, R_1 has a candidate key

$R_1 = ISQ$, $R_2 = SD$, $R_3 = IB$, $R_4 = BO$

decomposition of R into R_1, R_2, R_3 and R_4 is lossless and dependency preserving.

Q.12 *(a) Explain 3rd NF with suitable example.* *[R.T.U. 2017]*

(b) Explain BCNF with suitable example. *[R.T.U. 2017]*

Ans.(a) *Refer to Q.11.*

Ans.(b) *Refer to Q.11.*

Q.13 *Discuss the purpose of BCNF and describe how BCNF different from 3NF. Provide an example to illustrate your answer.* *[R.T.U. 2016, 2012]*

OR

Why BCNF to be considered stricter than 3NF? Explain decomposition of non-BCNF scheme into BCNF scheme. *[R.T.U. 2015]*

OR

Define BCNF. How does it differ from 3 NF? Why it is considered a stronger form of 3 NF? *[R.T.U. Dec. 2013, 2008]*

OR

Why BCNF considered to be stricter than 3NF? How is non BCNF scheme decomposed into BCNF scheme? *[R.T.U. 2011]*

OR

Discuss the purpose of BCNF and describe how BCNF different from 3NF. Provide an example to illustrate your answer. *[R.T.U. 2010; Raj. Univ. 2005, 2003]*

Ans.BCNF (Boyce-Code Normal Form) : *Refer to Q.11.*

Third Normal Form (3NF) : *Refer to Q.11.*

Comparison of BCNF and 3NF: Of the two normal forms for relational database schemas, 3NF and BCNF, there are advantages to 3NF in that we know that it is always possible to obtain a 3NF design without sacrificing a lossless-join or dependency preservation. Nevertheless, there are disadvantages to 3NF. If we do not eliminate all transitive relation schema dependencies, we may have to use null values to represent some of the possible meaningful relationship among data items and there is the problem of repetition of information.

The goals of database design with functional dependencies are :

1. BCNF
2. Lossless-join
3. Dependency preservation

Since it is not possible to satisfy all three, we may be forced to choose between BCNF and dependency preservation with 3NF.

BCNF Vs 3NF

BCNF is an extended form of 3NF. If a relation is BCNF then it must be in 3NF. In BCNF we extend our concept up to all the candidate keys of the relation, which are concatenated and two or more of the candidate share a common attribute.

"To be in BCNF, a table must only have candidate key as determinants".

Boyce - Code Normal Form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter form of 3NF, because every relation in BCNF is also in 3NF. The formal definition of BCNF differs slightly from the definition of 3NF. A relation schema R is in BCNF if whenever a nontrivial functional dependency $x \rightarrow A$ holds in R, then x is a super key of R.

Decomposition of Non-BCNF into BCNF :

One of the desirable normal form that eliminates all the redundancy which can be discovered and based on functional dependency is BCNF.

A relation schema R is in BCNF with aspect to a set F of functional dependency if, all the functional dependencies in f^+ of the form $\alpha \rightarrow \beta$ where

$\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds :

- $\alpha \rightarrow \beta$ is a trivial functional dependency (*i.e.*, $\beta \subseteq \alpha$)
- α is a super key for schema R

Example : Consider an example that is not in BCNF.

bar (–) loan = (Custid, loanno, amount). The functional dependency loan_no $\rightarrow$ amount holds on bar_loan, but loan_ number is not a super key (because a loan may be made to a consortium of many customers).

If we decompose the bar_loan relation into borrower and loan. Then, the borrower schema is in BCNF because no non-trivial functional dependency holds on it. The loan schema has one non-trivial functional dependency

that holds, loan_number → amount, but loan_no is a super key (in this primary key) for loan. Thus, loan is in BCNF.

Example of BCNF : *Refer to Q.11.*

Example : Suppose we have a database for an instrument firm, consisting of the following attributes : B (Broker), O (office of the Broker), I (Investor), S (stock), Q (Quantity of stock owned by an investor) and D (dividend paid by a stock), with the functional dependencies S → D, I → B, Is → Q and B → Q.

(i) Find a key for the relation schema R = BOSQID

Ans. Attribute closure

$S^+ = D$

$I^+ = BO$

$(IS)^+ = BOQDI$ S, contains all attributes

∴ (IS) is a key.

(ii) How many key does relation schema R have?

Ans. (IS) is the only key as described above in (i).

(iii) Find a lossless-join decomposition of R into Boyce-code Normal Form. Find a decomposition of R into third Normal Form having a lossless-join and preserving dependencies

Ans. **BCNF**

S → D is non-trivial and S is not a key

∴ R is not in BCNF. Decompose R

$R_1 = (R - \beta) = BOSQI$, $R_2 = (\alpha, \beta) = (S, D)$

$R_1 \cap R_2 \rightarrow R_2$, R_1: I → B is non-trivial. I is not a key. R_1 is not in BCNF, decompose R_1 into $R_3 = (R_1 - \beta) =$ OSQI and $R_4 = (\alpha, \beta) = (I, \beta)$

∴ Lossless decomposition of R into

R_2 = SD, R_3 = OSQI and R_4 = IB is in BCNF.

Q.14 ***(a) What is Bad database? Explain with examples insert, update and delete anomalies in database.***

(b) Consider the schema R = (A, B, C, D, E) with a set F of functional dependencies {ED→D, A→BC, E→A, E→B, B→C}. Find canonical cover for F. ***[R.T.U. 2014]***

Ans. (a) Bad Database : The purpose of database design is to arrange the corporate data fields into an organised structure such that it generates set of relationships and stores information without unnecessary redundancy. In fact, the redundancy and database consistency are the most important logical criteria in database design. A bad database design may result into repetitive data and information and an inability to represent desired information. It is, therefore, important to examine the relationships that exist among the data of an entity to refine the database design.

A poorly planned database may make it tougher to enter or update new records or it may allow many mistakes to get in. You also see this difference when you're trying to pull information from the database, which, of course, is the whole reason you're going to the trouble o creating and managing that database in the first place.

Customer Number : 1454834 **Customer** : W. Coyote General Delivery Falling Rocks, A2 84211 (599) 555-9345			**Terms** : Net 30 **Ship Via** : USPS **Order Date** : 21/10/2015	
Product No.	**Description**	**Quantity**	**Unit Price**	**Extended Amount**
SPR-2290	Super strength spring	2	24.00	$48.00
STR-67	Foot straps, leather	2	2.50	$5.00
HLM-45	Deluxe Crash Helmet	1	67.80	$67.88
KPR-1	Rocket, solid fuel	1	128, 200.40	$120,200.40
KLT-7	Emergency Location Transmitter	1	79.80	**Free Gift**
Total Amount				$128,321.28

Fig. : Invoice from acme industries

Insert Anomaly : The insert anomaly refers to a situation wherein one,cannot insert a new tuple into a relation because of an artificial dependency on another relation. The error that has caused the anomaly is that attributes of two different entities are mixed into the same relation. Referring to fig., we see that the ID, name, and address of the customer are included in the invoice view. Were we to merely make a relation from this view as it is, and eventually a table from the relation, we would soon discover that we could not insert a new customer into the database unless they had bought something. This is because all the customer data is embedded in the invoice.

Delete Anomaly : The delete anomaly is just the opposite of the insert anomaly. It refers to a situation wherein a deletion of data about one particular entity causes unintended loss of data that characterizes another entity. In the case of the acme industries invoice, if we delete the last invoice that belongs to a particular customer, we lose all the data related to that customer. Again, this is because data from two entities (customers and invoices) would be incorrectly mixed into a single relation if we merely implemented the invoice as a table without applying the normalization process to the relation.

Update Anomaly : The update anomaly refers to a situation where an update of a single data value requires multiple tuples (rows) of data to be updated. In our invoice example, if we wanted to change the customer's address,

we would have to change it on every single invoice for the customer. This is because the customer address would be redundantly stored in every invoice for the customer. To make matters worse, redundant data provides the golden opportunity to update many copies of the data, but miss a few of them, which results in inconsistent data. The mantra of the skilled database designer is, for each attribute, capture it once, store it once, and use that one copy everywhere.

Ans. (b) First combine $E \rightarrow A$ and $E \rightarrow B$ into $E \rightarrow AB$.

Set is now $\{ED \rightarrow D, A \rightarrow BC, E \rightarrow AB, B \rightarrow C\}$

C is extraneous in $A \rightarrow BC$

Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies.

Yes : using transitivity on $A \rightarrow B$ and $B \rightarrow C$.

Set is now $\{A \rightarrow B, B \rightarrow C, ED \rightarrow D, E \rightarrow AB\}$

B is extraneous in $E \rightarrow AB$

Check if $E \rightarrow B$ is logically implied by $E \rightarrow A$ and other dependencies.

Yes : using transitivity on $E \rightarrow A$ and $A \rightarrow B$.

Set is now $\{A \rightarrow B, B \rightarrow C, E \rightarrow A, ED \rightarrow D\}$

Thus, the canonical cover is

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

❑❑❑

TRANSACTION PROCESSING 4

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 ***What is the need of serializability in transaction processing?*** ***[R.T.U. 2019]***

OR

What is serializability?

Ans. Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order.

Q.2 ***What is concurrency?*** ***[R.T.U. 2019]***

OR

What is a concurrent execution of transaction?

Ans. Concurrent execution of transaction means multiple transactions execute/run concurrently in RDBMS with each transaction doing its atomic unit of work for the operations encapsulated in the particular transaction.

Q.3 ***What is cascadeless schedule?***

Ans. A cascadeless schedule (also known as recoverable schedule) is one where, for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti , the commit operation of Ti appears before the read operation of Tj .

Q.4 ***What is graphical form of recoverable schedule?***

Ans.

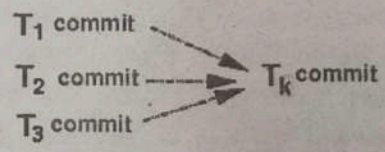
If:

T_1 $w_1(A)$ ------> $r_k(A)$

T_2 $w_2(B)$ ------> $r_k(B)$ T_k

T_3 $w_3(C)$ ------> $r_k(C)$

Then:



The transaction T_k ***must*** **commit** ***only after*** the transactions T_1, T_2 and T_3 have **committed.**

Q.5 ***What is use of serialization graph?***

Ans. Serialization graph is used to test the serializability of a schedule.

PART-B

Q.6 ***Write a short note on transaction properties and recoverable schedules.*** ***[R.T.U. 2019]***

Ans. Transaction Properties : The four basic properties of a Transaction Processing System are referred to as the ACID properties. Atomicity, Consistency, Isolation and Durability. These properties are used to test that a transaction is never unfinished, the data held in the system

is always consistent, concurrent transactions are independent and the effects of a transaction continue after the transaction is completed.

Atomicity : The transaction must happen completely or it should be undone completely. If a transaction fails, the effects of all operations that make up the transaction need to be cancelled and the data needs to be returned to its original previous state. If we look at the first part of the word "Atomicity", atom, the term suggests its meaning at the time the concept was first used. An atom is considered the smallest item of matter that can exist.

An example of how a transaction processing system handles atomicity is a student making a purchase from the school canteen. Money changes hands and the student later realizes that the drink is past its use by date. As there are no other drinks available, the student returns the drink and the canteen refunds the purchase price. This transaction is not recorded and so, no transaction occurred.

Consistency : For a successful transaction to take place, all parties must agree on the facts of the exchange. When a successful transaction is completed, the data should be in a consistent state. This means that all data should be able to be accounted for and that each step of the transaction is carried out in the same way each time. This ensures that data is correct for each part of the transaction.

An example of this property would see the sale of two drinks in the canteen should increase the takings by the canteen and decrease the stock held in the canteen by two drinks. The steps are to exchange the drinks for money. The two should balance so that the data is consistent.

Durability : The effects of a completed transaction should always be durable. In a large organization, transactions need to be recorded so that they can be checked at any time in the future. Backups need to be kept of this data. This is also part of an organizations responsibility to pay tax and meet other obligations.

Isolation : Transaction must be independent of each other. This doesn't actually happen. The user just believes that it does. Isolation involves the appearance of treating each transaction separately and keeping the data for each transaction separate.

The aim of isolation is to prevent the same data being treated twice. For example, an organization selling tickets to rock concerts such as Ticketed need to ensure they do not sell the same seat to more than one person. There may be many outlets selling tickets as well as the online booking website. Customers who access the online booking site need to be sure that their transaction is separate. It is vital that no one else can book the same seat through another agent. Isolation is a bigger issue as the Transaction Processing System becomes larger as it becomes difficult to make it appear as if each transaction is being handled sequentially.

Recoverable Schedule : *Refer to Q.3 and Q.4.*

Q.7 ***What is the states of transactions? Explain briefly.***

Ans. A transaction goes through many different states throughout its life cycle. These states are called as transaction states. Transaction states are as follows :

(i) Active state
(ii) Partially committed state
(iii) Committed state
(iv) Failed state
(v) Aborted state
(vi) Terminated state

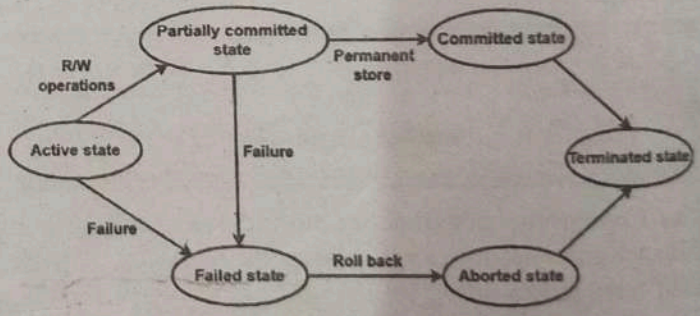


Fig. Transaction States in DBMS

(i) Active State : This is the first state in the life cycle of a transaction. A transaction is called in an active state as long as its instructions are getting executed. All the changes made by the transaction now are stored in the buffer in main memory.

(ii) Partially Committed State : After the last instruction of transaction has executed, it enters into a partially committed state. After entering this state, the transaction is considered to be partially committed. It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

(iii) Committed State: After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state. Now, the transaction is considered to be fully committed.

Note : After a transaction has entered the committed state; it is not possible to roll back the transaction. In other words, it is not possible to undo the changes that have been made by the transaction. This is because the system is updated into a new consistent state. The only way to undo the changes is by carrying out another transaction called as compensating transaction that performs the reverse operations.

(iv) Failed State : When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.

(v) Aborted State : After the transaction has failed and entered into a failed state, all the changes made by it have to be undone. To undo the changes made by the transaction, it becomes necessary to roll back the transaction. After the transaction has rolled back completely, it enters into an aborted state.

(vi) Terminated State : This is the last state in the life cycle of a transaction. After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.

Q.8 ***What is the properties of transaction processing?***

Ans. *Refer to Q.6.*

PART-C

Q.9 ***Explain conflict v/s view serializability in detail.***

[R.T.U. 2019]

Ans. Conflict Serializability : Instructions Ii and Ij, of transactions Ti and Tj respectively, conflict if and only if there exists some item P accessed by both Ii and Ij, and atleast one of these instructions wrote P.

Consider the below operations :

i. Ii = read(P), Ij = read(P). Ii and Ij don't conflict.

ii. Ii = read(P), Ij = write(P). They conflict.

iii. Ii = write(P), Ij = read(P). They conflict.

iv. Ii = write(P), Ij = write(P). They conflict.

A conflict between Ii and Ij forces a temporal order between them.

If Ii and Ij are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

If a schedule Scan be transformed in to a schedule S by a series of swaps of non-conflicting instructions, then S and S' are conflict equivalent.

In other words a schedule Sis conflict serializable if it is conflict equivalent to a serial schedule.

Example of a schedule that is not conflict serializable:

T3	T4
Read(P)	
	Write(P)
Write(P)	

The instructions cannot be swapped in the above schedule to obtain either the serial schedule < T 3 , T 4 >, or the serial schedule < T 4 ,T 3 >.

A serial schedule T2 follows T1, by a series of swaps of non-conflicting instructions making the below Schedule confict serializable.

T1	T2
Read(X)	
Write(X)	
	Read(X)
	Write(X)
Read(Y)	
Write(Y)	
	Read(Y)
	Write(Y)

View Serializability : S and S' are view equivalent if the following three conditions are met:

i. For each data item P, if transaction Ti reads the initial value of P in schedule S, then transaction Ti must, in schedule S', also read the initial value of P.

ii. For each data item P, if transaction Ti executes read (P)in schedule S, and that value was produced by transaction Tj, then transaction Ti must in

schedule S' also read the value of P that was produced by transaction Tj.

iii. For each data item P, the transaction that performs the final write(P) operation in schedule S must perform the final write(P) operation in schedule S'.

View equivalence is also based purely on reads and writes alone.

A schedule S is view serializable if it is ie w equivalent to a serial schedule.

Every conflict serializable schedule is also view serializable.

Every view serializable schedule which is not conflict serializable has blind writes.

T3	T4	T6
Read(P)		
	Write(P)	
Write(P)		
		Write(P)

Q.10 ***What is cascadeless schedule? Why is cascadeless ness of schedules desirable? Are there any circumstances under which it would be desirable to allow non-cascadeless schedules? Explain and justify your answer.***

[R.T.U. 2019]

Ans. Cascadeless schedule : *Refer to Q.3.*

Recoverability : A recoverable schedule is one where, for each pair of Transaction T_i and T_j such that T_j reads data item previously written by T_i the commit operation of T_i appears before the commit operation T_j.

T8	T9	
read(A)		T9 is dependent on T8
write(A)		Non recoverable schedule if T9 commits before T8
	read(A)	
read(B)		

Suppose that the system allows T9 to commit immediately after execution of read(A) instruction. Thus T9 commit before T8 does.

Now suppose that T8 fails before it commits. Since T9 has read the value of data item A written by T8 we must abort T9 to ensure transaction Atomicity.

However, T9 has already committed and cannot be aborted. Thus we have a situation where it is impossible to recover correctly from the failure of T8.

Cascadeless schedules

T10	T11	T12
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Transaction T10 writes a value of A that is read by Transaction T11. Transaction T11 writes a value of A that is read by Transaction T12. Suppose at this point T10 fails. T10 must be rolled back, since T11 is dependent on T10, T11 must be rolled back, T12 is dependent on T11, T12 must be rolled back.

This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks is called Cascading rollback.

- Cascading rollback is undesirable, since it leads to the undoing of a significant amount of work.
- It is desirable to restrict the schedules to those where cascading rollbacks cannot occur, Such schedules are called Cascadeless Schedules.
- Formally, a cascadeless schedule is one where for each pair of transaction T_i and T_j such that T_j reads data item, previously written by T_i the commit operation of T_i appears before the read operation of T_j.

Every Cascadeless schedule is also recoverable schedule.

Q.11 ***Describe conflict serializability and view serializability with examples?***

Ans. *Refer to Q.9.*

□□□

CONCURRENCY CONTROL 5

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 *How many kinds of locks present in lock-based protocols?*

Ans. Locks are of two types :

Binary Locks : A lock on a data item can be in two states; it is either locked or unlocked.

Shared/Exclusive : This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Q.2 *What is the definition of timestamp-based protocols?*

Ans. The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp. Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one. In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Q.3 *Write down the three phase present in the validation based protocol?*

Ans. In the validation based protocol, the transaction is executed in the following three phases:

(i) Read phase : In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.

(ii) Validation phase : In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.

(iii) Write phase : If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Q.4 *What is deadlock handling?*

Ans. Deadlock is a state of a database system having two or more transactions, when each transaction is waiting for a data item that is being locked by some other

transaction. A deadlock can be indicated by a cycle in the wait-for-graph. This is a directed graph in which the vertices denote transactions and the edges denote waits for data items.

PART-B

Q.5 ***What is shadow paging? Explain in detail.***

[R.T.U. 2019]

Ans. In computer science, shadow paging is a technique for providing atomicity and durability (two of the ACID properties) in database systems. A page in this context refers to a unit of physical storage (probably on a hard disk), typically of the order of 1 to 64 KB. Shadow paging is a copy-on-write technique for avoiding in-place updates of pages. Instead, when a page is to be modified, a shadow page is allocated. Since the shadow page has no references (from other pages on disk), it can be modified liberally, without concern for consistency constraints, etc. When the page is ready to become durable, all pages that referred to the original are updated to refer to the new replacement page instead. Because the page is "activated" only when it is ready, it is atomic.

If the referring pages must also be updated via shadow paging, this procedure may recur many times, becoming quite costly. One solution, employed by the WAFL file system (Write Anywhere File Layout) is to be lazy about making pages durable (i.e. write-behind caching). This increases performance significantly by avoiding many writes on hotspots high up in the referential hierarchy (e.g.: a file system superblock) at the cost of high commit latency.

Write-ahead logging is a more popular solution that uses in-place updates. Shadow paging is similar to the old master–new master batch processing technique used in mainframe database systems. In these systems, the output of each batch run (possibly a day's work) was written to two separate disks or other form of storage medium. One was kept for backup, and the other was used as the starting point for the next day's work. Shadow paging is also similar to purely functional data structures, in that in-place updates are avoided.

Q.6 ***How many types are present in database failures?***

Ans. There are many types of failures that can affect database processing. Some failures affect the main memory only, while others involve secondary storage. Following are the types of failure:

Hardware Failures: Hardware failures may include memory errors, disk crashes, bad disk sectors, disk full errors and so on. Hardware failures can also be attributed to design errors, inadequate (poor) quality control during fabrication, overloading (use of under-capacity components) and wear out of mechanical parts.

Software Failures: Software failures may include failures related to software's such as, operating system, DBMS software, application programs and so on.

It's the one call no database administrator wants to receive: There's been a database failure. Wherever they are, whatever they are doing, they must drop everything and head to the office ASAP. The company's whole livelihood relies on their ability to get the data back and get the systems up and running as quickly as possible. And while everyone dreams of the day they can be their company's savior, in the case of database failures, a true hero is the one that never receives that dreaded call.

There are several types of database failure. While there's always a risk, with some time and care, you can protect your company from some of the most common types of database failure.

Three Types of Database Failures

1. System Crash: Mayday indeed. When the system crashes, it's a race towards reinstating affected processes or – worse yet – data recovery.

A system crash usually refers to any kind of bugs or hardware malfunction in the operating system or the database software. It can bring the processing of transaction to a halt and can even cause the loss of content residing on volatile storage such as main memory, cache memory, RAM, etc.

There are many reasons why database system might crash. Maintaining strict security and maintenance routines and protocols in the database, its host system, and the network should guarantee minimal downtime. In the event something did happen, the DBA must have a

documented crisis plan to restore and reinstate the database as quickly as possible.

2. Media Failure : This one is very risky. Media failures are caused by a head crash or unreadable media. These types of data failures are considered one of the most serious because it is possible for entire data loss. Media failures usually leave database systems unavailable for several hours until recovery is complete, especially in applications with large devices and high transaction volume.

The best way to prevent this type of database failure is to protect your data with adequate malware protection and backing up your data frequently.

3. Application Software Errors : When the resource limit is exceeded, bad input, logical or internal errors occur, or any other factors related to the application software is compromised, transactions can fail giving way to database failure.

It is generally recommended that application software errors are minimized in the software code during conception and the software engineering process. It is better for developers to put mechanisms and controls into place during the design of the architecture and coding operation, than trying to remedy mistakes later. Asides from failures, malicious code may exploit known vulnerabilities, especially those associated with a particular programming software tool or known human software coding error.

Q.7 ***What is the technique of log-based recovery? Explain briefly in detail?***

Ans. Atomicity property of DBMS states that either all the operations of transactions must be performed or none. The modifications done by an aborted transaction should not be visible to database and the modifications done by committed transaction should be visible.

To achieve our goal of atomicity, user must first output to stable storage information describing the modifications, without modifying the database itself. This information can help us ensure that all modifications performed by committed transactions are reflected in the database. This information can also help us ensure that no modifications made by an aborted transaction persist in the database.

Log and log records : The log is a sequence of log records, recording all the update activities in the database. In a stable storage, logs for each transaction are maintained. Any operation which is performed on the database is recorded is on the log. Prior to performing any modification to database, an update log record is created to reflect that modification.

An update log record represented as: <Ti, Xj, V1, V2> has these fields:

Transaction identifier: Unique Identifier of the transaction that performed the write operation.

- **Data item :** Unique identifier of the data item written.
- **Old value :** Value of data item prior to write.
- **New value :** Value of data item after write operation.

Other type of log records is:

<Ti start> : It contains information about when a transaction Ti starts.

<Ti commit> : It contains information about when a transaction Ti commits.

<Ti abort> : It contains information about when a transaction Ti aborts.

Undo and Redo Operations : Because all database modifications must be preceded by creation of log record, the system has available both the old value prior to modification of data item and new value that is to be written for data item. This allows system to perform redo and undo operations as appropriate:

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

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

The database can be modified using two approaches :

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

(ii) Immediate Modification Technique : If database modification occurs while transaction is still active, it is said to use immediate modification technique.

Recovery using Log records : After a system crash has occurred, the system consults the log to determine which transactions need to be redone and which need to be undone.

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

Transaction Ti needs to be redone if log contains record <Ti start> and either the record <Ti commit> or the record <Ti abort>.

Use of Checkpoints : When a system crash occurs, user must consult the log. In principle, that needs to search the entire log to determine this information. There are two major difficulties with this approach:

The search process is time-consuming.

Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will cause recovery to take longer.

To reduce these types of overhead, user introduce checkpoints. A log record of the form <checkpoint L> is used to represent a checkpoint in log where L is a list of transactions active at the time of the checkpoint. When a checkpoint log record is added to log all the transactions that have committed before this checkpoint have <T commit> log record before the checkpoint record. Any database modifications made by Ti are written to the database either prior to the checkpoint or as part of the checkpoint itself. Thus, at recovery time, there is no need to perform a redo operation on Ti.

After a system crash has occurred, the system examines the log to find the last <checkpoint L> record. The redo or undo operations need to be applied only to transactions in L, and to all transactions that started execution after the record was written to the log. Let us denote this set of transactions as T. Same rules of undo and redo are applicable on T as mentioned in Recovery using Log records part.

Note that user need to only examine the part of the log starting with the last checkpoint log record to find the set of transactions T, and to find out whether a commit or abort record occurs in the log for each transaction in T. For example, consider the set of transactions {T0, T1, . . ., T100}. Suppose that the most recent checkpoint took place during the execution of transaction T67 and T69, while T68 and all transactions with subscripts lower than 67 completed before the checkpoint. Thus, only transactions T67, T69, . . ., T100 need to be considered during the recovery scheme. Each of them needs to be redone if it has completed (that is, either committed or aborted); otherwise, it was incomplete, and needs to be undone.

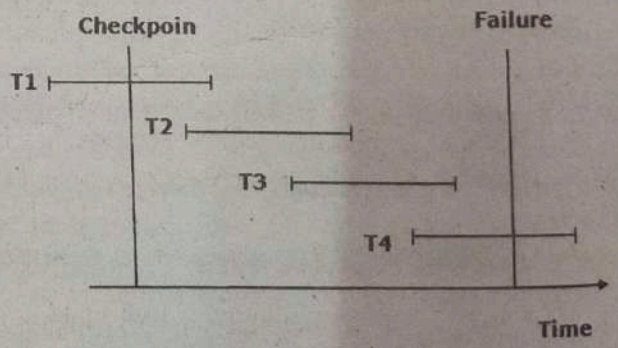
PART-C

Q.8 ***Why must lock and unlock be atomic operations? Explain recovery related data structure in detail. Also explain deadlock handling.***

[R.T.U. 2019]

Ans. Lock and unlock must be atomic operations because otherwise it may be possible for two transactions to obtain an exclusive lock on the same object, thereby destroying the principles of 2PL. A lock is held over a long duration, and a latch is released immediately after the physical read or writes operation is completed. Convoy is a queue of waiting transactions. It occurs when a transaction holding a heavily used lock is suspended by the operating system, and every other transactions that needs this lock is queued.

Recovery using Checkpoint : In the following manner, a recovery system recovers the database from this failure:



The recovery system reads log files from the end to start. It reads log files from T4 to T1.

Recovery system maintains two lists, a redo-list, and an undo-list.

The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or

just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.

For example: In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.

The transaction is put into undo state if the recovery system sees a log with <Tn, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.

For example: Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

Deadlock Handling : *Refer to Q.4.*

Q.9 ***Why use concurrent transaction with recovery process? Explain recovery using with checkpoint?***

Ans. Whenever more than one transaction is being executed, then the interleaved of logs occur. During recovery, it would become difficult for the recovery system to backtrack all logs and then start recovering. To ease this situation, 'checkpoint' concept is used by most DBMS. As we have discussed checkpoint in Transaction Processing Concept of this tutorial, so you can go through the concepts again to make things more clear.

Checkpoint : The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk. The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.

When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.

The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery using Checkpoint : *Refer to Q.8.*

Q.10 ***Defined the whole concurrency control types and examples?***

Ans. In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories.

- Lock based protocols
- Time stamp based protocols

Lock-based Protocols: Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds.

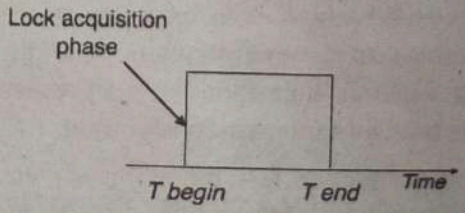
Binary Locks : A lock on a data item can be in two states; it is either locked or unlocked.

Shared/exclusive : This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

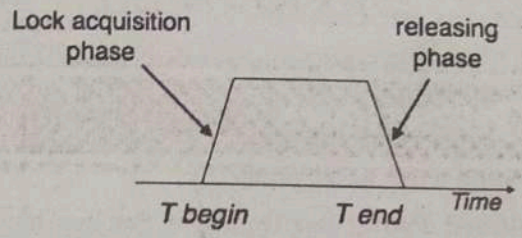
There are four types of lock protocols :

Simplistic Lock Protocol: Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

Pre-claiming Lock Protocol: Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.

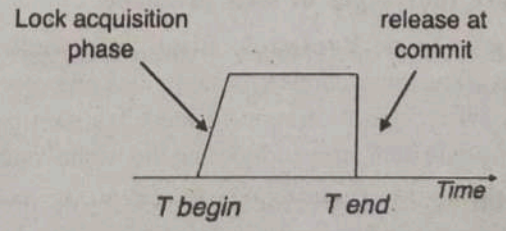


Two-Phase Locking 2PL: This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.



Two-phase locking has two phases, one is growing, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released. To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking: The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols: The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one. In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Timestamp Ordering Protocol : The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and writes 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 Ti is denoted as TS(Ti).

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 Ti is rolled back.

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

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

□□□