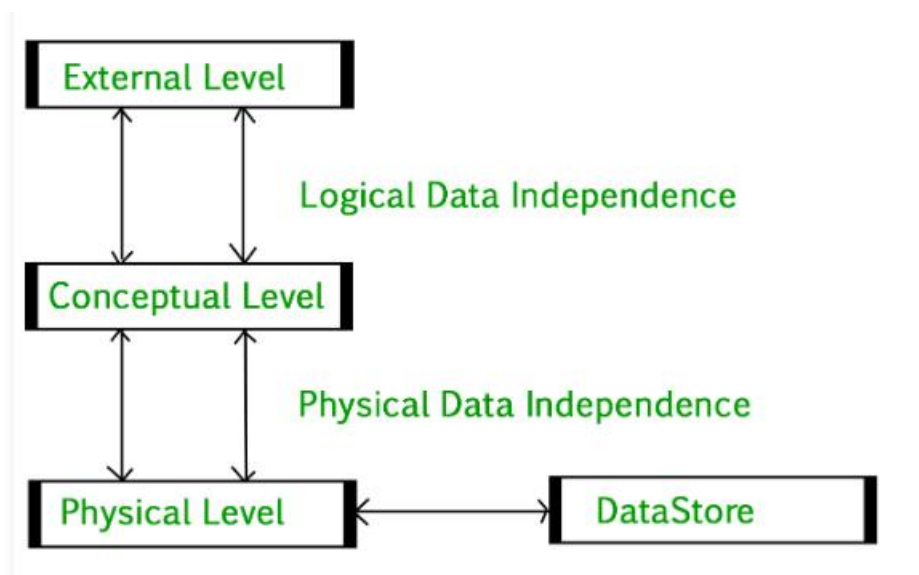


DBMS ALL ANSWERS

1(A). Briefly explain the three tier architecture for the data independence

Ans:

The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced. DBMS 3-tier architecture divides the complete system into three inter-related but independent



1. **Physical Level:** At the physical level, the information about the location of database objects in the data store is kept. Various users of DBMS are unaware of the locations of these objects. In simple terms, physical level of a database describes how the data is being stored in secondary storage devices like disks and tapes and also gives insights on additional storage details.
2. **Conceptual Level:** At conceptual level, data is represented in the form of various database tables. For Example, STUDENT database may contain STUDENT and COURSE tables which will be visible to users but users are unaware of their storage. Also referred as logical schema, it describes what kind of data is to be stored in the database.
3. **External Level:** An external level specifies a view of the data in terms of conceptual level tables. Each external level view is used to cater to the needs of a particular category of users. For Example, FACULTY of a university is interested in looking course details of students, STUDENTS are interested in looking at all details related to academics, accounts, courses and hostel details as well. So,

different views can be generated for different users. The main focus of external level is data abstraction.

**** Data independence means a change of data at one level should not affect another level. Two types of data independence are present in this architecture:

1. **Physical Data Independence:** Any change in the physical location of tables and indexes should not affect the conceptual level or external view of data. This data independence is easy to achieve and implemented by most of the DBMS.
2. **Conceptual Data Independence:** The data at conceptual level schema and external level schema must be independent. This means a change in conceptual schema should not affect external schema. e.g.; Adding or deleting attributes of a table should not affect the user's view of the table. But this type of independence is difficult to achieve as compared to physical data independence because the changes in conceptual schema are reflected in the user's view.

1(B). Explain advantages and disadvantages of DBMS over File System

Ans:

DBMS	File System
DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	File system is a collection of data. In this system, the user has to write the procedures for managing the database.
DBMS gives an abstract view of data that hides the details.	File system provides the detail of the data representation and storage of data.
DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure.	File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost.

DBMS provides a good protection mechanism.	It is very difficult to protect a file under the file system.
DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	File system can't efficiently store and retrieve the data.
DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information.

2(A). Explain the concept of Specialization, generalization and aggregation in E_R diagrams. Give one example for each one of them

Ans:

Generalization

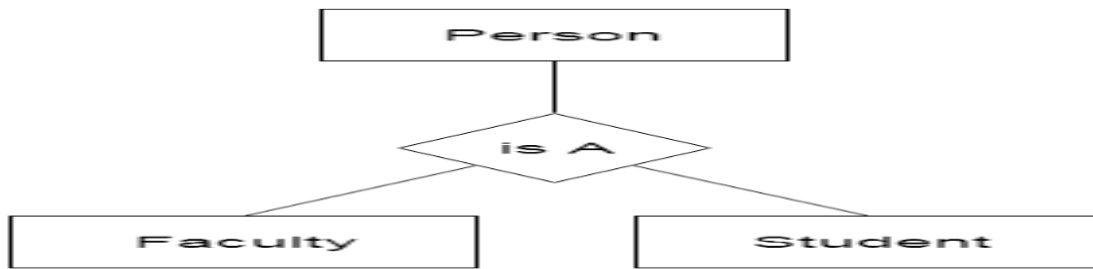
Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.

In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.

Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.

In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.

For example, Faculty and Student entities can be generalized and create a higher level entity Person.



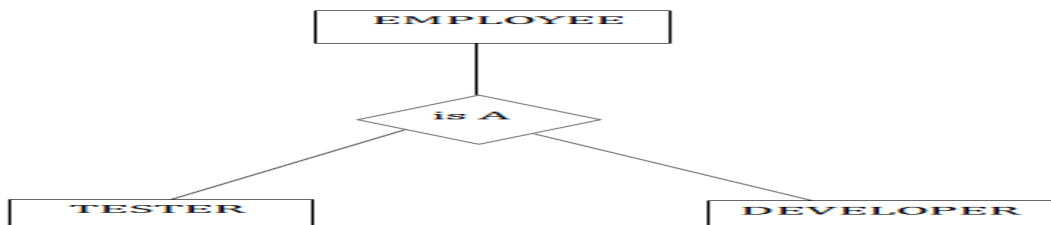
Specialization

Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.

Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.

Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

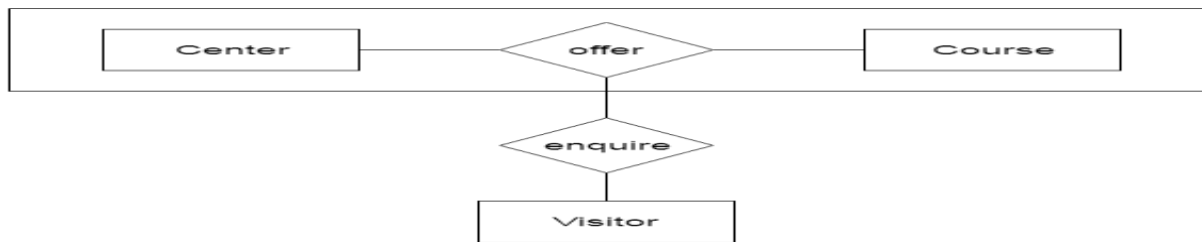
For example: In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

For example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.



2(B). Explain centralized and client server architecture for the database

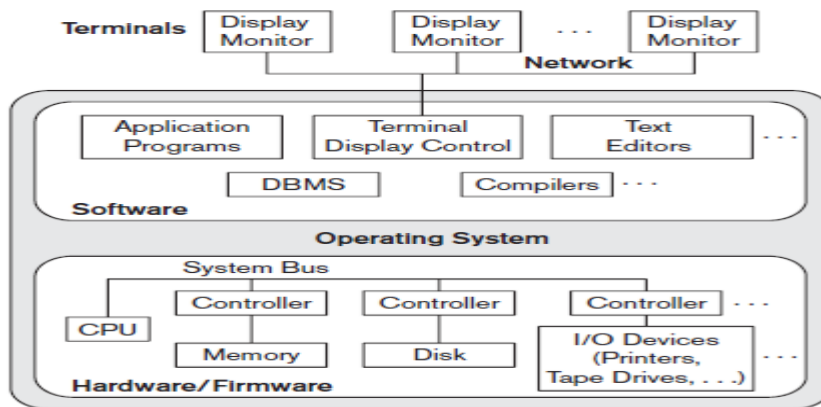
Ans:

Centralized and Client/Server Architecture for DBMS

Centralized Architecture for DBMS:

In Centralized Architecture, the mainframe computers are used for processing all system functions including User application Programs and User Interface Programs as well as DBMS functionalities. This is because in earlier days, most users accessed such systems via Computer Terminals, which can't Process and they have only display capability. Therefore the Processing used to takeplace in these Computer Systems and the display information is sent to display terminals and these terminals are connected to mainframe computers via various kinds of Networks.

As the days pass by, we are now having PersonalComputers(PC's) in the market. But still in the beginning Centralized Architecture for DBMS was used. Gradually the DBMS systems started to make use of Processing Power in the used side i.e Computers have come with Processing Power and in turn led to the use of Client/Server Architecture.



Centralized Architecture for DBMS

Client/Server Architecture:

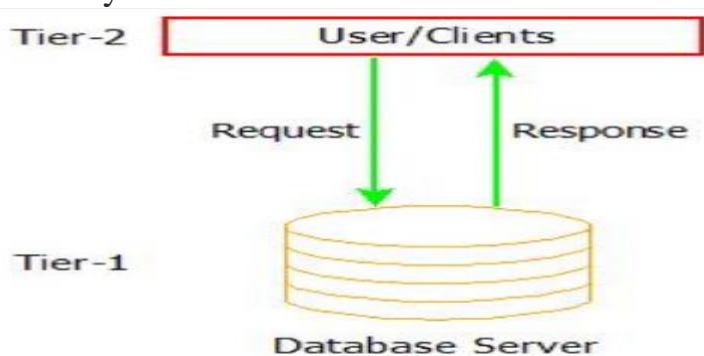
The concept of client/server architecture assumes an underlying framework that consists of many PCs as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks. A client in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality, such as database access which does not exist at that machine, it connects to a server that provides the needed functionality.

A Server is a system which contains both Hardware and Software which provides services to client Machines like file access, printing and database access.

1) Two Tier Client/Server Architecture for DBMS: Here Two-tier means that our Architecture has two layers, which are client layer and Datalayer. In Client layer we have several Client machines which can have the access to the database server. The API present on the client machine will establish the connection between the machine and the Database server through JDBC something else. This is because Clients and Database Server may be at different different

locations. Once this connection gets established, the Interface present on the client machine contains an Application Program on the back-side which contains a query. This query will be processed by the Database server and in turn the queried information will be sent to the client machine.

For example if we query the database to retrieve some information, the query will be Processed by Database server and that information will be sent to the client by Database server itself!!!



Two-Tier Architecture for DBMS

2) Three-Tier client/server Architecture for DBMS: Here there is an additional layer which acts as an intermediate between Client layer and Datalayer called Business logic layer. Business logic layer is the layer where the Application Programs are processed. Here the Application Programs are processed in the Application server itself, which makes it different from Two-tier Architecture where queries are processed in the database server. Simply the Client machines will contact Application Server which in turn processes our Application Programs and fetches the Required Data from Database and then sends this Information back to the client machine in the suitable format only.



Three-Tier client/server Architecture for DBMS:- Now we may think that Two-Tier Architecture is easy to use and maintain and why we should go for

Three-Tier. The Reason is Three-Tier Architecture is Scalable and more secured. Even it is easy to maintain Two-Tier Architecture of DBMS it is still not scalable when we have large number of clients and also not secure because the clients are having direct access to database server. But Three-Tier Architecture ensures Scalability and Security of the data because of the presence of this Intermediate layer which processes the queries and it just retrieves data from server instead of processing in the server to take place.

3(A). Define and discuss the functions of Database Administrator (DBA)

Ans:

Database administration is more of an operational or technical level function responsible for physical database design, security enforcement, and database performance. Tasks include maintaining the data dictionary, monitoring performance, and enforcing organizational standards and security.

These are the functions of a data administrator (not to be confused with database administrator functions):

1. Data policies, procedures, standards
2. Planning- development of organization's IT strategy, enterprise model, cost/benefit model, design of database environment, and administration plan.
3. Data conflict (ownership) resolution
4. Data analysis- Define and model data requirements, business rules, operational requirements, and maintain corporate data dictionary
5. Internal marketing of DA concepts
6. Managing the data repository

The DBA is responsible for many critical tasks:

Design of the conceptual and physical schemas: The DBA is responsible for interacting with the users of the system to understand what data is to be stored in the DBMS and how it is likely to be used. Based on this knowledge, the DBA must design the conceptual schema (decide what relations to store) and the physical schema (decide how to store them). The DBA may also design widely

used portions of the external schema, although users will probably augment this schema by creating additional views.

Security and authorization: The DBA is responsible for ensuring that unauthorized data access is not permitted. In general, not everyone should be able to access all the data. In a relational DBMS, users can be granted permission to access only certain views and relations. For example, although you might allow students to find out course enrollments and who teaches a given course, you would not want students to see faculty salaries or each others' grade information.

Data availability and recovery from failures: The DBA must take steps to ensure that if the system fails, users can continue to access as much of the uncorrupted data as possible. The DBA must also work to restore the data to a consistent state. The DBMS provides software support for these functions, but the DBA is responsible for implementing procedures to back up the data periodically and to maintain logs of system activity (to facilitate recovery from a crash).

Database tuning: The needs of users are likely to evolve with time. The DBA is responsible for modifying the database, in particular the conceptual and physical schemas, to ensure adequate performance as user requirements change.

3(B). List out various database applications and explain them.

Ans:

Applications of DBMS

In so many fields, we will use a database management system.

Let's see some of the applications where database management system uses –

- **Railway Reservation System** – The railway reservation system database plays a very important role by keeping record of ticket booking, train's departure time and arrival status and also gives information regarding train late to people through the database.
- **Library Management System** – Now-a-days it's become easy in the Library to track each book and maintain it because of the database. This happens because there are thousands of books in the library. It is very difficult to keep a record of all books in a copy or register. Now DBMS used to maintain all the information related to book issue dates, name of the book, author and availability of the book.
- **Banking** – Banking is one of the main applications of databases. We all know there will be a thousand transactions through banks daily and we are

doing this without going to the bank. This is all possible just because of DBMS that manages all the bank transactions.

- **Universities and colleges** – Now-a-days examinations are done online. So, the universities and colleges are maintaining DBMS to store Student's registrations details, results, courses and grade all the information in the database. For example, telecommunications. Without DBMS there is no telecommunication company. DBMS is most useful to these companies to store the call details and monthly postpaid bills.
- **Credit card transactions** – The purchase of items and transactions of credit cards are made possible only by DBMS. A credit card holder has to know the importance of their information that all are secured through DBMS.
- **Social Media Sites** – By filling the required details we are able to access social media platforms. Many users sign up daily on social websites such as Facebook, Pinterest and Instagram. All the information related to the users are stored and maintained with the help of DBMS.
- **Finance** – Now-a-days there are lots of things to do with finance like storing sales, holding information and finance statement management etc. these all can be done with database systems.
- **Military** – In military areas the DBMS is playing a vital role. Military keeps records of soldiers and it has so many files that should be kept secure and safe. DBMS provides a high security to military information.
- **Online Shopping** – Now-a-days we all do Online shopping without wasting the time by going shopping with the help of DBMS. The products are added and sold only with the help of DBMS like Purchase information, invoice bills and payment.
- **Human Resource Management** – The management keeps records of each employee's salary, tax and work through DBMS.
- **Manufacturing** – Manufacturing companies make products and sell them on a daily basis. To keep records of all those details DBMS is used.
- **Airline Reservation system** – Just like the railway reservation system, airlines also need DBMS to keep records of flights arrival, departure and delay status.

So finally, we can clearly conclude that the DBMS is playing a very important role in each and every field.

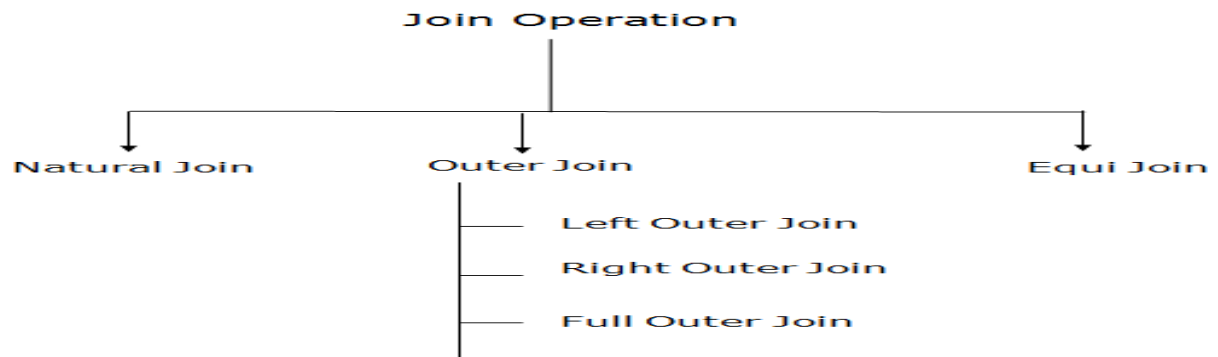
4(A). Classify different join operations (Relational Algebra & SQL) and explain with example.

Ans:

Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by \bowtie . Example:

Operation: (EMPLOYEE \bowtie SALARY)



Joins in Relational Algebra

1. Natural Join:

A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.

It is denoted by \bowtie .

Example: Let's use the above EMPLOYEE table and SALARY table:

\bowtie EMP_NAME, SALARY (EMPLOYEE \bowtie SALARY)

2. Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example: (EMPLOYEE \bowtie FACT_WORKERS)

An outer join is basically of three types:

1. Left outer join
2. Right outer join
3. Full outer join

a. Left outer join:

Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.

In the left outer join, tuples in R have no matching tuples in S. It is denoted by \bowtie .

Example: Using the above EMPLOYEE table and FACT_WORKERS table

EMPLOYEE \bowtie FACT_WORKERS

b. Right outer join:

Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.

In right outer join, tuples in S have no matching tuples in R. It is denoted by $\bowtie\lt$.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation

EMPLOYEE $\bowtie\lt$ FACT_WORKERS

c. Full outer join:

Full outer join is like a left or right join except that it contains all rows from both tables.

In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name. It is denoted by $\bowtie\lt\gt$.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

EMPLOYEE $\bowtie\lt\gt$ FACT_WORKERS

3. Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example: CUSTOMER \bowtie PRODUCT

SQL Joins

A **JOIN** clause is used to combine rows from two or more tables, based on a related column between them.

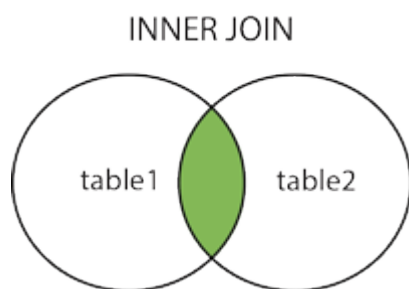
example :

we can create the following SQL statement (that contains an **INNER JOIN**), that selects records that have matching values in both tables: Orders, Customers

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

SQL INNER JOIN Keyword

The **INNER JOIN** keyword selects records that have matching values in both tables.



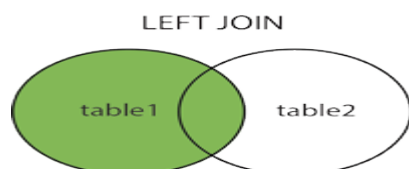
The following SQL statement selects all orders with customer information:

Example

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

SQL LEFT JOIN Keyword

The **LEFT JOIN** keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.



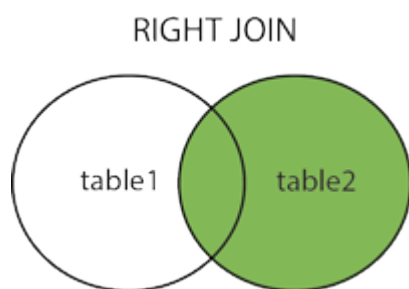
The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;
```

SQL RIGHT JOIN Keyword

The **RIGHT JOIN** keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.



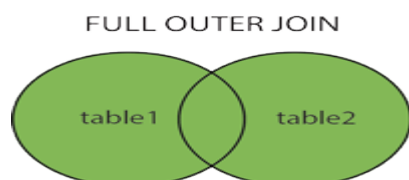
The following SQL statement will return all employees, and any orders they might have placed:

Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
ORDER BY Orders.OrderID;
```

SQL FULL OUTER JOIN Keyword

The **FULL OUTER JOIN** keyword returns all records when there is a match in left (table1) or right (table2) table records.



The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

SQL Self Join

A self join is a regular join, but the table is joined with itself.

The following SQL statement matches customers that are from the same city:

Example

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

4(B). Explain following operators in Relational Algebra i) selection ii) projection iii) rename

Ans:

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

1. Select Operation:

The select operation selects tuples that satisfy a given predicate. It is denoted by sigma (σ). Notation: $\sigma p(r)$

Where:

- σ is used for selection prediction
 r is used for relation
 p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like $=, \neq, \geq, <, >, \leq$.

Example :

σ BRANCH_NAME="perryride" (LOAN)

2. Project Operation:

This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.

It is denoted by π . Notation: $\pi A_1, A_2, A_n (r)$

Where **A1, A2, A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

π NAME, CITY (CUSTOMER)

3. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by ρ (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

ρ (STUDENT1, STUDENT)

4. Define Relational Model & Explain the concept of domain, attribute, tuple, relation with an example?

Ans:

Relational Model concept

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Domain: It contains a set of atomic values that an attribute can take.

Attribute: It contains the name of a column in a particular table. Each attribute A_i must have a domain, $\text{dom}(A_i)$

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Tuple – It is nothing but a single row of a table, which contains a single record.

Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence

Example: STUDENT Relation

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27

Ganesh	17282	9028 9i3988	Delhi	40
--------	-------	-------------	-------	----

In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.

The instance of schema STUDENT has 5 tuples.

t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

5(A). What is view of data? Explain the three levels of data independence.

Ans:

View of data in DBMS describes the abstraction of data at three-level i.e. **physical level, logical level, view level**. The physical level of abstraction defines how data is stored in the storage and also reveals its access path

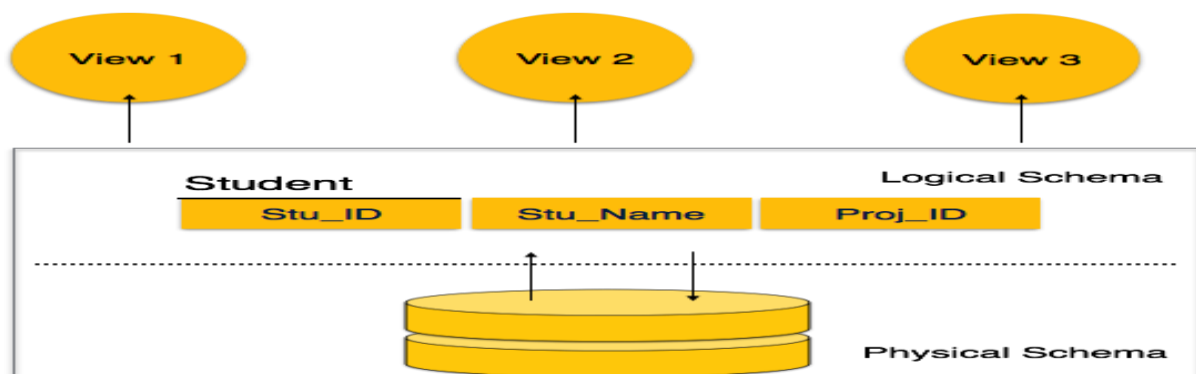
Data Abstraction is a process of hiding unwanted or irrelevant details from the end user.

It provides a different **view** and helps in achieving data independence which is used to enhance the security of data.

The database systems consist of complicated data structures and relations. For users to access the data easily, these complications are kept hidden, and only the relevant part of the database is made accessible to the users through data abstraction.

Mainly there are three levels of abstraction for DBMS, which are as follows –

- Physical or Internal Level
- Logical or Conceptual Level
- View or External Level



Physical or Internal Level

It is the lowest level of abstraction for DBMS which defines how the data is actually stored, it defines data-structures to store data and access methods used by the database. Actually, it is decided by developers or database application programmers how to store the data in the database.

So, overall, the entire database is described in this level that is physical or internal level. It is a very complex level to understand. For example, customer's information is stored in tables and data is stored in the form of blocks of storage such as bytes, gigabytes etc.

Logical or Conceptual Level

Logical level is the intermediate level or next higher level. It describes what data is stored in the database and what relationship exists among those data. It tries to describe the entire or whole data because it describes what tables to be created and what are the links among those tables that are created.

It is less complex than the physical level. Logical level is used by developers or database administrators (DBA). So, overall, the logical level contains tables (fields and attributes) and relationships among table attributes.

View or External Level

It is the highest level. In view level, there are different levels of views and every view only defines a part of the entire data. It also simplifies interaction with the user and it provides many views or multiple views of the same database.

View level can be used by all users (all levels' users). This level is the least complex and easy to understand.

For example, a user can interact with a system using GUI that is view level and can enter details at GUI or screen and the user does not know how data is stored and what data is stored, this detail is hidden from the user.

Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

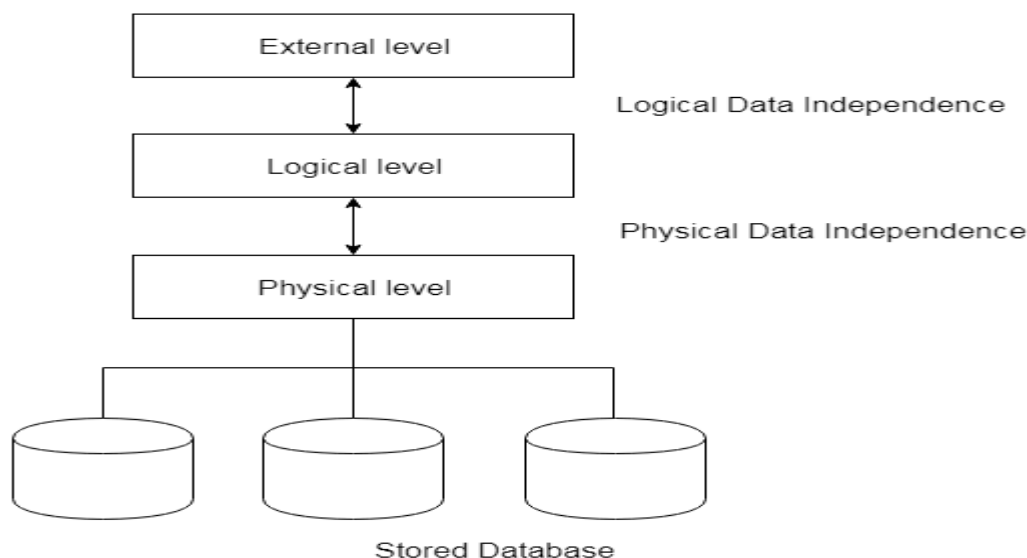
There are two types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.



5(B). Explain Commit, Rollback and Save point commands in SQL with suitable examples

Ans:

TCL Commands in SQL

In SQL, TCL stands for Transaction control language.

A single unit of work in a database is formed after the consecutive execution of commands is known as a transaction.

There are certain commands present in SQL known as TCL commands that help the user manage the transactions that take place in a database.

COMMIT, ROLLBACK and SAVEPOINT are the most commonly used TCL commands in SQL.

1. COMMIT

COMMIT command in SQL is used to save all the transaction-related changes permanently to the disk.

Syntax: COMMIT;

Example

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example which would delete those records from the table which have age = 25 and then COMMIT the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
      WHERE AGE = 25;
SQL> COMMIT;
```

Thus, two rows from the table would be deleted and the SELECT statement would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00

3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

2. SAVEPOINT

We can divide the database operations into parts. For example, we can consider all the insert related queries that we will execute consecutively as one part of the transaction and the delete command as the other part of the transaction. Using the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names. *For example*, we can save all the insert related queries with the savepoint named INS. To save all the insert related queries in one savepoint, we have to execute the SAVEPOINT query followed by the savepoint name after finishing the insert command execution.

Syntax:

SAVEPOINT savepoint_name;

Example

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

The following code block contains the series of operations.

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.
SQL> SAVEPOINT SP2;
```

```

Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted.
SQL> SAVEPOINT SP3;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.

```

Now that the three deletions have taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone –

```

SQL> ROLLBACK TO SP2;
Rollback complete.

```

Notice that only the first deletion took place since you rolled back to SP2.

```

SQL> SELECT * FROM CUSTOMERS;
+---+-----+-----+-----+-----+
| ID | NAME   | AGE | ADDRESS | SALARY |
+---+-----+-----+-----+-----+
| 2 | Khilan | 25 | Delhi   | 1500.00 |
| 3 | kaushik | 23 | Kota    | 2000.00 |
| 4 | Chaitali | 25 | Mumbai  | 6500.00 |
| 5 | Hardik | 27 | Bhopal  | 8500.00 |
| 6 | Komal | 22 | MP      | 4500.00 |
| 7 | Muffy | 24 | Indore  | 10000.00 |
+---+-----+-----+-----+-----+
6 rows selected.

```

3. ROLLBACK

While carrying a transaction, we must create savepoints to save different parts of the transaction. According to the user's changing requirements, he/she can roll back the transaction to different savepoints. *Consider a scenario:* We have initiated a transaction followed by the table creation and record insertion into the table. After inserting records, we have created a savepoint INS. Then we executed a delete query, but later we thought that mistakenly we had removed the useful record. Therefore in such situations, we have an option of rolling back our transaction. In this case, we have to roll back our transaction using the **ROLLBACK** command to the savepoint INS, which we have created before executing the DELETE query.

Syntax:

ROLLBACK TO savepoint_name;

Example

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example, which would delete those records from the table which have the age = 25 and then ROLLBACK the changes in the database.

```
SQL> DELETE FROM CUSTOMERS  
WHERE AGE = 25;  
SQL> ROLLBACK;
```

Thus, the delete operation would not impact the table and the SELECT statement would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

5(C). What is Relation? Differentiate between a relation Schema and Relation Instance

Ans:

A relationship in a DBMS, is **primarily the way two or more data sets are linked**. This is so true for Relational Database Management Systems. One dataset may be then termed as the Foreign key and the ones linked to it may be termed as the Primary Key. There may be multiple Foreign and Primary keys linked to each other.

Relational Model concept

Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

example: STUDENT

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 9i3988	Delhi	40

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

A relational schema is an outline that shows how companies store and organize information within a database. It also shows what connections make up the database. Developers often view relational schemas as the shape, blueprint or design of the sets of information within the database. Relational schemas do not contain actual data, however, because it is simply a blueprint. A developer's

goal is to design the schema in a way that the information is readable and avoids redundancy. The developer can choose to display a schema as a visual depiction, like a graph, or as formulas written in coding language.

ex : STUDENT(name,roll_no,phone_no,address,age)

6. Discuss in detail about PL/SQL Procedures with examples

Ans:

PL/SQL Concepts

What is PL/SQL

PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. PL/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7). The functionalities of PL/SQL usually extended after each release of Oracle database. Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

Block Structure

- PL/SQL blocks have a pre-defined structure in which the code is to be grouped. Below are different sections of PL/SQL blocks.

1. Declaration section

2. Execution section

3. Exception-Handling section

syntax of pl/sql structure :

DECLARE –optional

<declarations>

BEGIN --mandatory

<executable statements. At least one executable statement is mandatory>

EXCEPTION --optional

<exception handles>

END; --mandatory

/

Example of PL/SQL constant

Let's take an example to explain it well:

DECLARE

-- constant declaration

pi constant number := 3.141592654;

-- other declarations

radius number(5,2); dia number(5,2); circumference number(7, 2); ar
ea number (10, 2);

BEGIN

-- processing

radius := 9.5;

dia := radius * 2; circumference := 2.0 * pi * radius; area := pi * radius *
radius;

-- output

dbms_output.put_line('Radius: ' || radius);

dbms_output.put_line('Diameter: ' || dia);

dbms_output.put_line('Circumference: ' || circumference);

dbms_output.put_line('Area: ' || area);

END;

/

After the execution of the above code at SQL prompt, it will produce the following result:.

Radius: 9.5

Diameter: 19

Circumference: 59.69

Area: 283.53

PL/SQL **procedure** successfully completed.

PL/SQL L

7. Discuss about Nested queries with an example.

Ans:

NESTED QUERIES

A nested query is a query that has another query embedded within it; the embedded query is called a subquery.

SQL subqueries are a powerful tool. They allow us to perform tasks more efficiently by having only one query instead of several.

When using nested queries, **keep these considerations in mind:**

- Subqueries can return **single values or tables** (with one or many rows and columns).
- You can include a subquery:
 - In the **WHERE** clause, to filter data.
 - In the **FROM** clause, to specify a new table.
 - In the **SELECT** clause, to specify a certain column.
 - In the **HAVING** clause, as a group selector.
- Subqueries should always be enclosed in parentheses ().
- Different database management systems have certain **limitations** on the number of subquery levels (e.g. up to 32 levels in SQL Server). However, in practice, you'll rarely have more than 2-3 levels of nested queries.
- Subqueries are often **computationally inefficient**. Thus, I recommend avoiding nested queries when other options are available (e.g. JOINS).

As an example, let us rewrite the following query, which we discussed earlier, using a nested subquery:

(Q1) Find the names of sailors who have reserved boat 103.

```
SELECT S.sname
FROM   Sailors S
WHERE  S.sid IN ( SELECT R.sid
                  FROM   Reserves R
                  WHERE  R.bid = 103 )
```

(Q2) Find the names of sailors who have reserved a red boat.

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN ( SELECT R.sid
                   FROM    Reserves R
                   WHERE   R.bid IN ( SELECT B.bid
                                     FROM   Boats B
                                     WHERE  B.color = 'red' ) )
```

*(Q21) Find the names of sailors who have **not** reserved a red boat.*

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid NOT IN ( SELECT R.sid
                      FROM    Reserves R
                      WHERE   R.bid IN ( SELECT B.bid
                                         FROM   Boats B
                                         WHERE  B.color = 'red' ) )
```

First of all, you can put a nested SELECT **within the WHERE clause** with comparison operators or the **IN**, **NOT IN**, **ANY**, or **ALL** operators. The second group of operators are used when your subquery returns a list of values (rather than a single value, as in the previous example):

- The **IN** operator checks if a certain value **is in the table** returned by the subquery.
- The **NOT IN** operator filters out the rows corresponding to the values **not present** in that table returned by a subquery.
- The **ANY** operator is used with comparison operators to **evaluate if any of the values** returned by the subquery satisfy the condition.
- The **ALL** operator is also used with comparison operators to **evaluate if all values** returned by the subquery satisfy the condition.

8. Discuss about different types of aggregate operators in SQL with

examples?

Ans:

AGGREGATE OPERATORS

We now consider a powerful class of constructs for computing *aggregate values* such as MIN and SUM. These features represent a significant extension of relational algebra. SQL supports five aggregate operations, which can be applied on any column, say A, of a relation:

1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
4. MAX (A): The maximum value in the A column.
5. MIN (A): The minimum value in the A column.

1. COUNT()

COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.

COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax : COUNT(*) or COUNT([ALL|DISTINCT] expression)

ex: SELECT COUNT(*) FROM PRODUCT_MAST;

Output: 10

Example: COUNT with WHERE

SELECT COUNT(*) FROM PRODUCT_MAST WHERE RATE>=20;

Output: 7

Example: COUNT() with GROUP BY

SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY;

Output:

Com1 5

Com2 3

Com3 2

Example: COUNT() with HAVING

SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY HAVING COUNT(*)>2;

Output:

Com1 5

Com2 3

2. SUM()

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax : SUM() or SUM([ALL|DISTINCT] expression)

Example: SUM()

SELECT SUM(COST) FROM PRODUCT_MAST;

Output:

670

3. AVG ()

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG() or AVG([ALL|DISTINCT] expression)

Example:

SELECT AVG(COST) FROM PRODUCT_MAST;

Output:

67.00

4. MAX()

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax : MAX() or MAX([ALL|DISTINCT] expression)

Example:

SELECT MAX(RATE) FROM PRODUCT_MAST;

Output :

30 5.

5. MIN()

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax : MIN() or MIN([ALL|DISTINCT] expression)

Example:

SELECT MIN(RATE) FROM PRODUCT_MAST;

Output: 10

9. Explain the term stored procedure and give examples why stored procedure are useful

Ans:- PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

Header: The header contains the name of the procedure and the parameters or variables passed to the procedure.

Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

How to pass parameters in procedure:

When you want to create a procedure or function, you have to define parameters. There are three ways to pass parameters in procedure:

IN parameters: The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.

OUT parameters: The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.

INOUT parameters: The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

A procedure may or may not return any value.

Create procedure example

In this example, we are going to insert record in user table. So you need to create user table first.

Table creation:

```
create table user(id number(10) primary key,name varchar2(100));
```

Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER" (id IN NUMBER, name IN  
VARCHAR2)
```

```
is
```

```
begin
```

```
insert into user values(id,name);
```

```
end;
```

```
/
```

Output:

Procedure created.

A stored procedure **provides an important layer of security between the user interface and the database**. It supports security through data access controls because end users may enter or change data, but do not write procedures.

The main advantages of stored procedure are given below:

- Better Performance – The procedure calls are quick and efficient as stored procedures are compiled once and stored in executable form.
- Higher Productivity
- Ease of Use
- Scalability
- Maintainability
- Security

10. Define functional dependency? Briefly explain about first normal form by taking suitable table.

Ans: Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

$$1. X \rightarrow Y$$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

11. Define Normalization? List out the problems caused by redundancy and explain them briefly. Illustrate fourth normal form with a suitable example

Ans: Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

Redundancy means having multiple copies of same data in the database. This problem arises when a database is not normalized. Suppose a table of student details attributes are: student Id, student name, college name, college rank, course opted.

Student_ID	Name	Contact	College	Course	R
100	Himanshu	7300934851	GEU	Btech	
101	Ankit	7900734858	GEU	Btech	
102	Aysuh	7300936759	GEU	Btech	
103	Ravi	7300901556	GEU	Btech	

As it can be observed that values of attribute college name, college rank, course is being repeated which can lead to problems. Problems caused due to redundancy are: Insertion anomaly, Deletion anomaly, and Updation anomaly.

1. Insertion Anomaly –

If a student detail has to be inserted whose course is not being decided yet then insertion will not be possible till the time course is decided for student.

Student_ID	Name	Contact	College	Course
100	Himanshu	7300934851	GEU	

This problem happens when the insertion of a data record is not possible without adding some additional unrelated data to the record.

2. Deletion Anomaly –

If the details of students in this table are deleted then the details of college will also get deleted which should not occur by common sense.

This anomaly happens when deletion of a data record results in losing some unrelated information that was stored as part of the record that was deleted from a table.

It is not possible to delete some information without losing some other information in the table as well.

3. Updation Anomaly –

Suppose if the rank of the college changes then changes will have to be all over the database which will be time-consuming and computationally costly.

Student_ID	Name	Contact	College	Course	Rank
100	Himanshu	7300934851	GEU	Btech	1
101	Ankit	7900734858	GEU	Btech	1
102	Aysuh	7300936759	GEU	Btech	1
103	Ravi	7300901556	GEU	Btech	1



Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency $A \twoheadrightarrow B$, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer

21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

12.What is meant by multi-valued dependency? Explain with suitable example.

Ans: Multivalued Dependency

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

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

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

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

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

1. BIKE_MODEL \twoheadrightarrow MANUF_YEAR
2. BIKE_MODEL \twoheadrightarrow COLOR

13.Explain in detail about 3NF and BCNF. List the differences between 3NF and BCNF

Ans: 1. [Third Normal Form \(3NF\)](#) :

A relation is said to be in Third Normal Form (3NF), if it is in [2NF](#) and when no non key attribute is transitively dependent on the primary key i.e., there is no transitive dependency. Also it should satisfy one of the below given conditions. For the function dependency C->D:

- C should be a super key and,
- D should be a prime attribute i.e, D should be a part of the candidate key.

3NF is used to reduce data duplication and to attain data integrity.

Example:

For the relation R(L, M, N, O, P) with functional dependencies as {L->M, MN->P, PO->L}:

The candidate keys will be : {LNO, MNO, NOP}

as the closure of LNO = {L, M, N, O, P}

closure of MNO = {L, M, N, O, P}

closure of NOP = {L, M, N, O, P}

This relation is in 3NF as it is already in 2NF and has no transitive dependency. Also there is no non prime attribute that is deriving a non prime attribute.

2. Boyce-Codd Normal Form (BCNF) :

BCNF stands for Boyce-Codd normal form and was made by R.F Boyce and E.F Codd in 1974. A functional dependency is said to be in BCNF if these properties hold:

- It should already be in 3NF.
- For a functional dependency say $P \rightarrow Q$, P should be a super key.

BCNF is an extension of 3NF and it is has more strict rules than 3NF. Also, it is considered to be more stronger than 3NF.

Example:

for the relation R(A, B, C, D) with functional dependencies as { $A \rightarrow B$, $A \rightarrow C$, $C \rightarrow D$, $C \rightarrow A$ }:

The candidate keys will be : {A, C}

as the closure of A = {A, B, C, D}

closure of C = {A, B, C, D}

This relation is in BCNF as it is already in 3NF (there is no prime attribute deriving no prime attribute) and on the left hand side of the functional dependency there is a candidate key.

Difference b/w 3NF and BCNF

S.NO.	3NF	BCNF
1.	In 3NF there should be no transitive dependency that is no non prime attribute should be transitively dependent on the candidate key.	In BCNF for any relation $A \rightarrow B$, A should be a super key of relation.
2.	It is less stronger than BCNF.	It is comparatively more stronger than 3NF.
3.	In 3NF the functional dependencies are already in 1NF and 2NF.	In BCNF the functional dependencies are already in 1NF, 2NF and 3NF.
4.	The redundancy is high in 3NF.	The redundancy is comparatively low in BCNF.
5.	In 3NF there is preservation of all functional dependencies.	In BCNF there may or may not be preservation of all functional dependencies.
6.	It is comparatively easier to achieve.	It is difficult to achieve.
7.	Lossless decomposition can be achieved by 3NF.	Lossless decomposition is hard to achieve in BCNF.

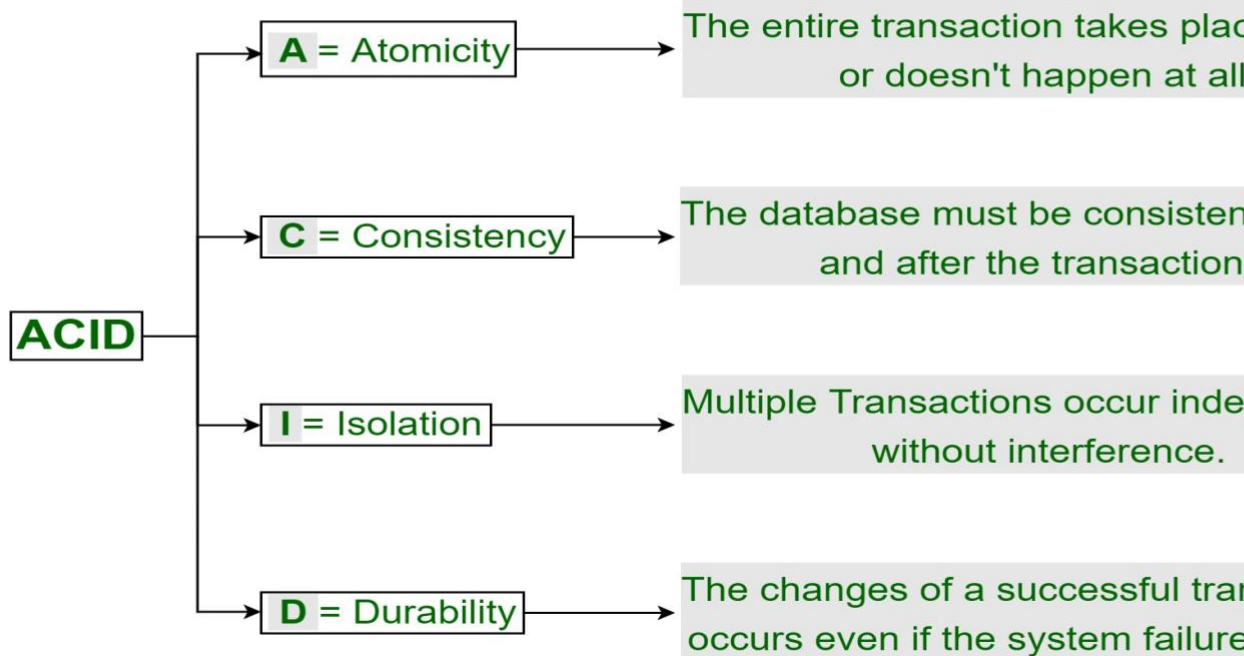
14A) Discuss the ACID properties. Illustrate them through examples?

Ans: ACID Properties in DBMS

A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

ACID Properties in DBMS



Atomicity:

By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

—**Abort**: If a transaction aborts, changes made to the database are not visible.

—**Commit**: If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X - 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**. (say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

Consistency:

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above, The total amount before and after the transaction must be maintained.
 Total **before T** occurs = **500 + 200 = 700**.
 Total **after T** occurs = **400 + 300 = 700**.
 Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Let **X** = 500, **Y** = 500.

Consider two transactions **T** and **T''**.

T	T''
Read (X)	Read (X)
X: = X*100	Read (Y)
Write (X)	Z: = X + Y
Read (Y)	Write (Z)
Y: = Y - 50	
Write (Y)	

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result, interleaving of operations takes place due to which **T''** reads the correct value of **X** but the incorrect value of **Y** and sum computed by

T'': (X+Y = 50, 000+500=50, 500)

is thus not consistent with the sum at end of the transaction:

T: (X+Y = 50, 000 + 450 = 50, 450).

This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

Some important points:

Property	Responsibility for maintaining properties
----------	---

Atomicity	Transaction Manager
-----------	---------------------

Consistency	Application programmer
-------------	------------------------

Isolation	Concurrency Control Manager
-----------	-----------------------------

Durability	Recovery Manager
------------	------------------

14B) Demonstrate about Transaction Management and different States of Transaction.

Ans: Transaction Management

Transactions in [DBMS](#):

Transactions are a set of operations used to perform a logical set of work. A transaction usually means that the data in the database has changed. One of the major uses of DBMS is to protect the user's data from system failures. It is done by ensuring that all the data is restored to a consistent state when the computer is restarted after a crash. The transaction is any one execution of the user program in a DBMS. Executing the same program multiple times will generate multiple transactions.

Example –

Transaction to be performed to withdraw cash from an ATM vestibule.

Set of Operations :

Consider the following example for transaction operations as follows.

Example -ATM transaction steps.

- Transaction Start.

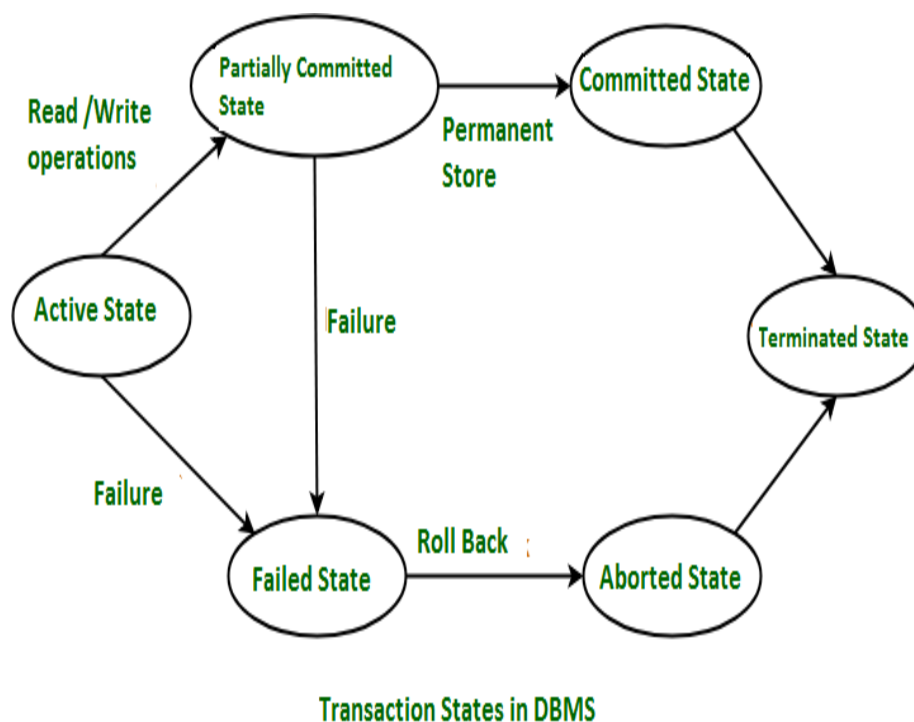
- Insert your ATM card.
- Select language for your transaction.
- Select Savings Account option.
- Enter the amount you want to withdraw.

Three operations can be performed in a transaction as follows.

1. Read/Access data (R).
2. Write/Change data (W).
3. Comm

Transaction States :

Transactions can be implemented using SQL queries and Server. In the below-given diagram, you can see how transaction states works.



Disadvantage of using a Transaction :

- It may be difficult to change the information within the transaction database by end-users.
- We need to always roll back and start from the beginning rather than continue from the previous state.

15A) Explain about conflict serializability and view serializability with suitable example

Ans: Conflict Serializable Schedule

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

Conflicting Operations

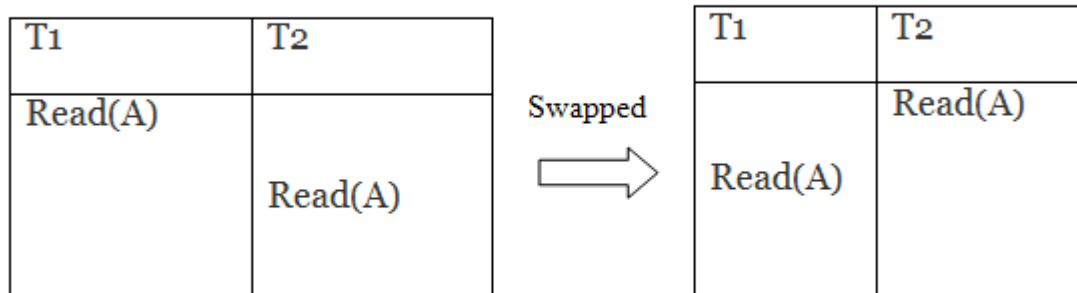
The two operations become conflicting if all conditions satisfy:

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

Example:

Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A) T2: Read(A)



Schedule S1

Schedule S2

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.

- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	Write(A)

Schedule S1

T1	T2
Read(A)	Write(A)

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule S1

T1	T2	T3
Write(A)	Write(A)	<u>Read(A)</u>

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S1

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A) Write(A)	Write(A)	Write(A)

Schedule S

With 3 transactions, the total number of possible schedule

1. $= 3! = 6$
2. $S1 = \langle T1 \ T2 \ T3 \rangle$
3. $S2 = \langle T1 \ T3 \ T2 \rangle$
4. $S3 = \langle T2 \ T3 \ T1 \rangle$
5. $S4 = \langle T2 \ T1 \ T3 \rangle$
6. $S5 = \langle T3 \ T1 \ T2 \rangle$
7. $S6 = \langle T3 \ T2 \ T1 \rangle$

Taking first schedule S1:

T1	T2	T3
Read(A) Write(A)	Write(A)	Write(A)

Schedule S1

Step 1: final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

15B) Define the term Lock. Explain various types of locks along with their compatibility functions.

Ans: Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

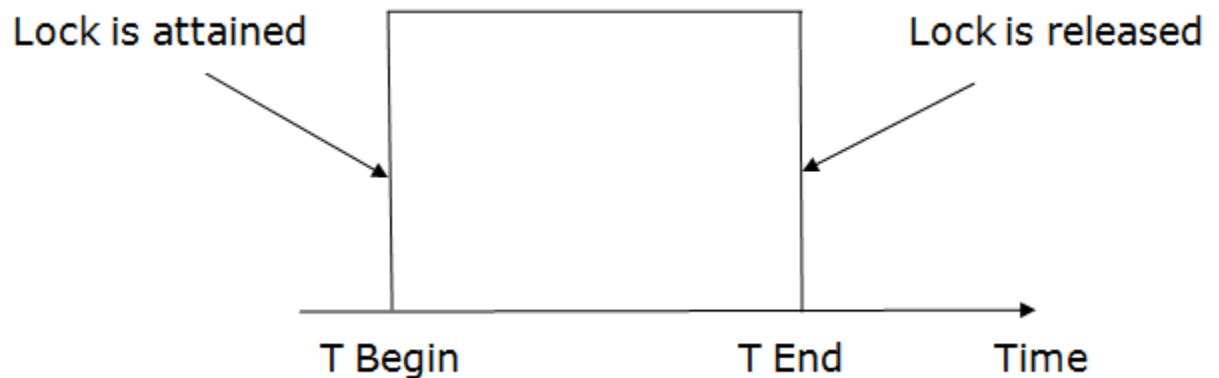
There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

2. Pre-claiming Lock Protocol

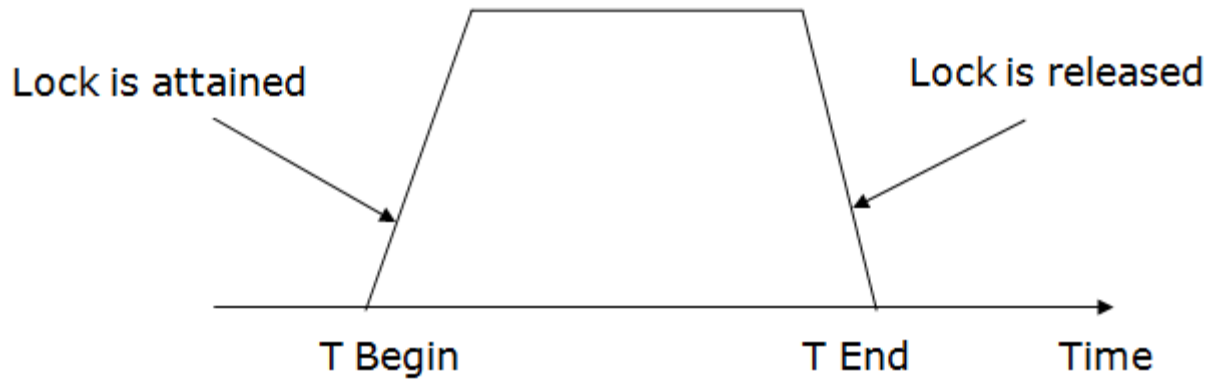
- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.

- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Example:

	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

The following way shows how unlocking and locking work with 2-PL.

Transaction T1:

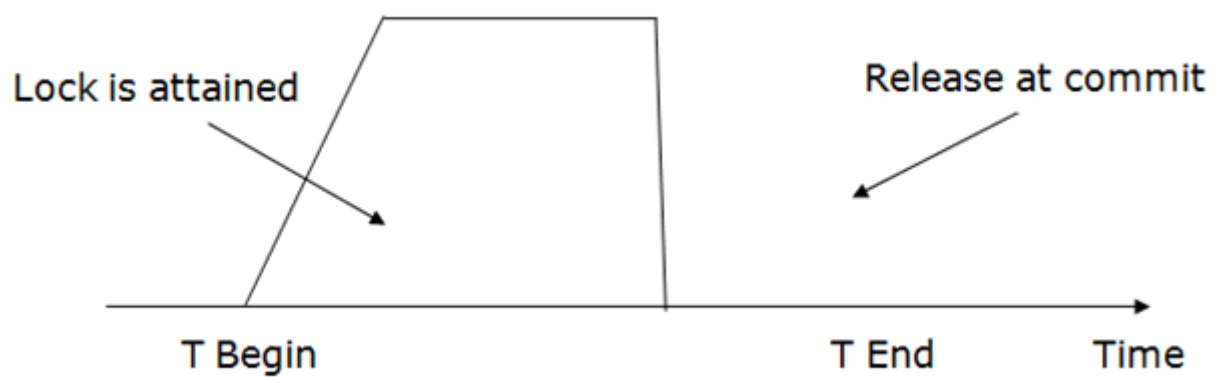
- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

Transaction T2:

- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

4. Strict Two-phase locking (Strict-2PL)

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.



It does not have cascading abort as 2PL does.

16. Explain concurrency control with Time-Stamp based protocols with suitable example

Ans: Timestamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.

- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction T_i issues a **Read(X)** operation:

- If $W_TS(X) > TS(T_i)$ then the operation is rejected.
- If $W_TS(X) \leq TS(T_i)$ then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction T_i issues a **Write(X)** operation:

- If $TS(T_i) < R_TS(X)$ then the operation is rejected.
- If $TS(T_i) < W_TS(X)$ then the operation is rejected and T_i is rolled back otherwise the operation is executed.

Where,

$TS(T_i)$ denotes the timestamp of the transaction T_i .

$R_TS(X)$ denotes the Read time-stamp of data-item X.

$W_TS(X)$ denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:



Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade- free.

17A) What is file organization? Explain in detail about types of file organizations

Ans: File Organization

- The **File** is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
- File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
- File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
- The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
- Files of fixed length records are easier to implement than the files of variable length records.

Objective of file organization

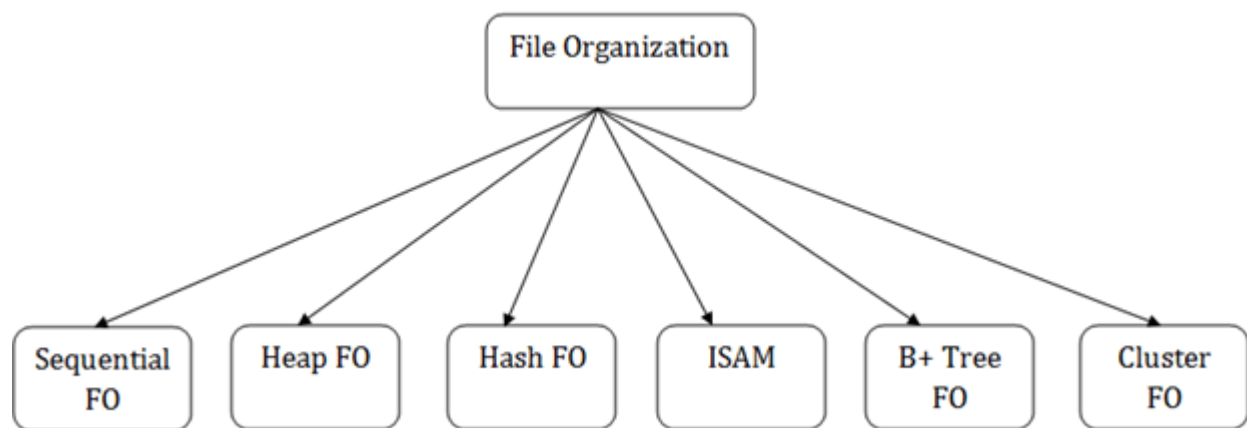
- It contains an optimal selection of records, i.e., records can be selected as fast as possible.

- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.
- For the minimal cost of storage, records should be stored efficiently.

Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:



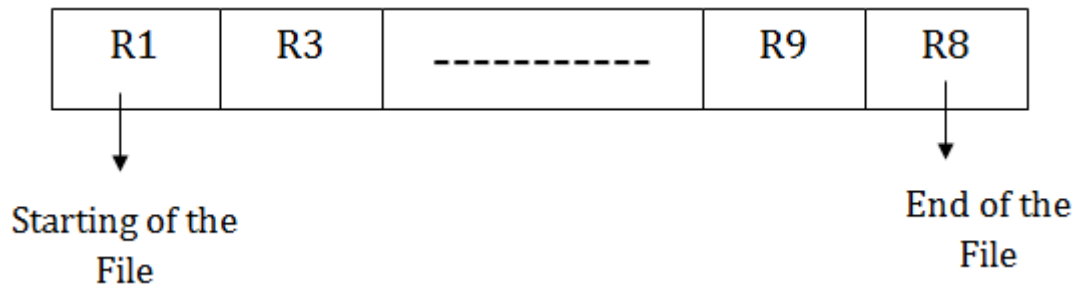
- [Sequential file organization](#)
- [Heap file organization](#)
- [Hash file organization](#)
- [B+ file organization](#)
- [Indexed sequential access method \(ISAM\)](#)
- [Cluster file organization](#)

Sequential File Organization

This method is the easiest method for file organization. In this method, files are stored sequentially. This method can be implemented in two ways:

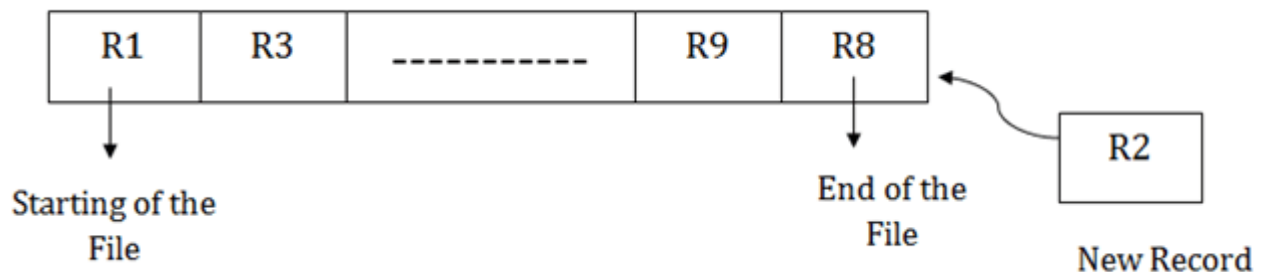
1. Pile File Method:

- It is a quite simple method. In this method, we store the record in a sequence, i.e., one after another. Here, the record will be inserted in the order in which they are inserted into tables.
- In case of updating or deleting of any record, the record will be searched in the memory blocks. When it is found, then it will be marked for deleting, and the new record is inserted.



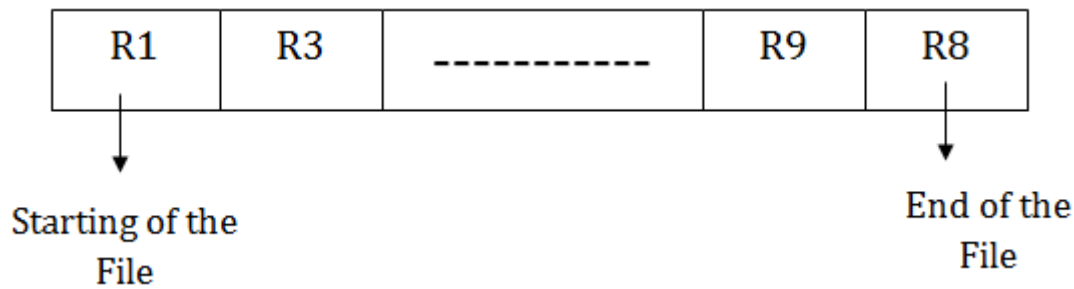
Insertion of the new record:

Suppose we have four records R1, R3 and so on upto R9 and R8 in a sequence. Hence, records are nothing but a row in the table. Suppose we want to insert a new record R2 in the sequence, then it will be placed at the end of the file. Here, records are nothing but a row in any table.



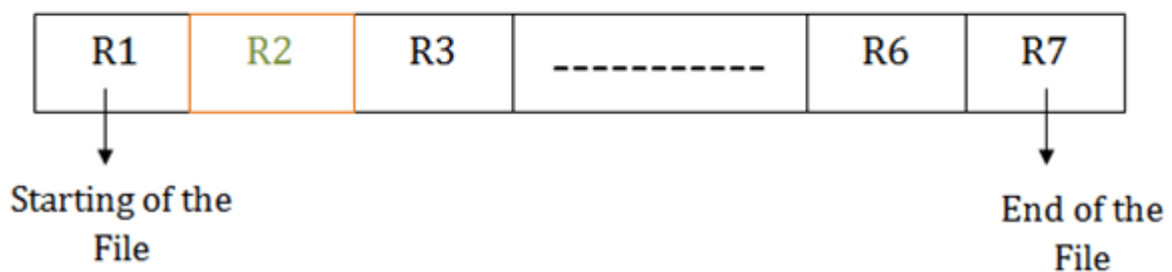
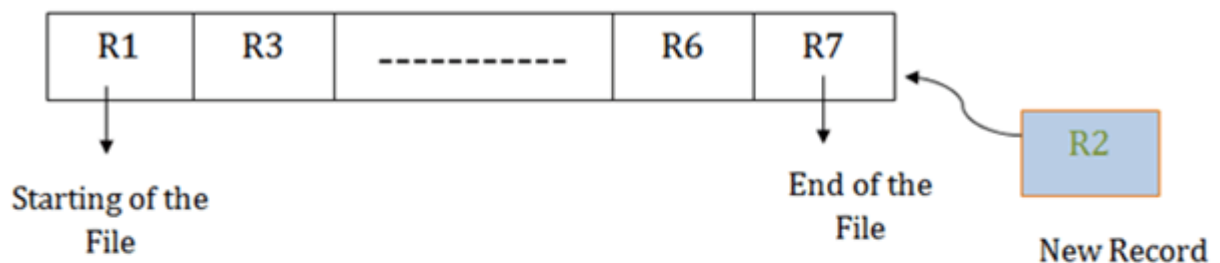
2. Sorted File Method:

- In this method, the new record is always inserted at the file's end, and then it will sort the sequence in ascending or descending order. Sorting of records is based on any primary key or any other key.
- In the case of modification of any record, it will update the record and then sort the file, and lastly, the updated record is placed in the right place.



Insertion of the new record:

Suppose there is a preexisting sorted sequence of four records R1, R3 and so on upto R6 and R7. Suppose a new record R2 has to be inserted in the sequence, then it will be inserted at the end of the file, and then it will sort the sequence.



Pros of sequential file organization

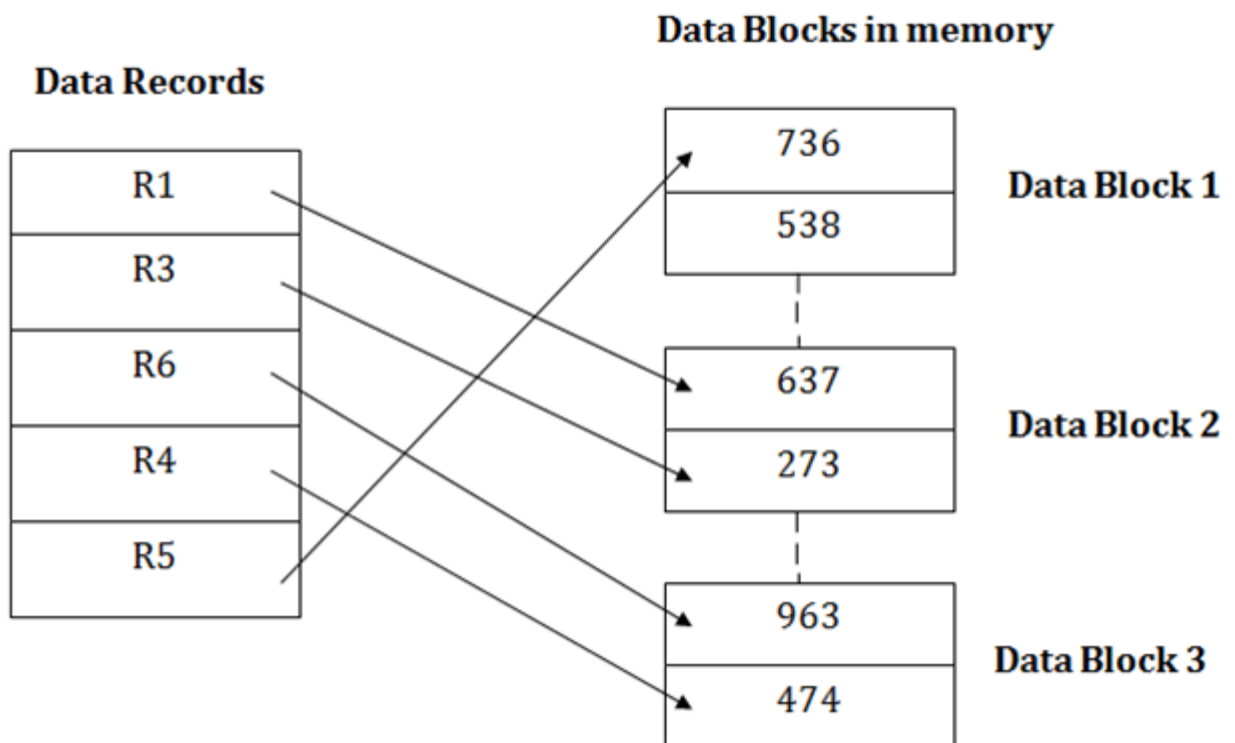
- It contains a fast and efficient method for the huge amount of data.
- In this method, files can be easily stored in cheaper storage mechanism like magnetic tapes.
- It is simple in design. It requires no much effort to store the data.
- This method is used when most of the records have to be accessed like grade calculation of a student, generating the salary slip, etc.
- This method is used for report generation or statistical calculations.

Cons of sequential file organization

- It will waste time as we cannot jump on a particular record that is required but we have to move sequentially which takes our time.
- Sorted file method takes more time and space for sorting the records.

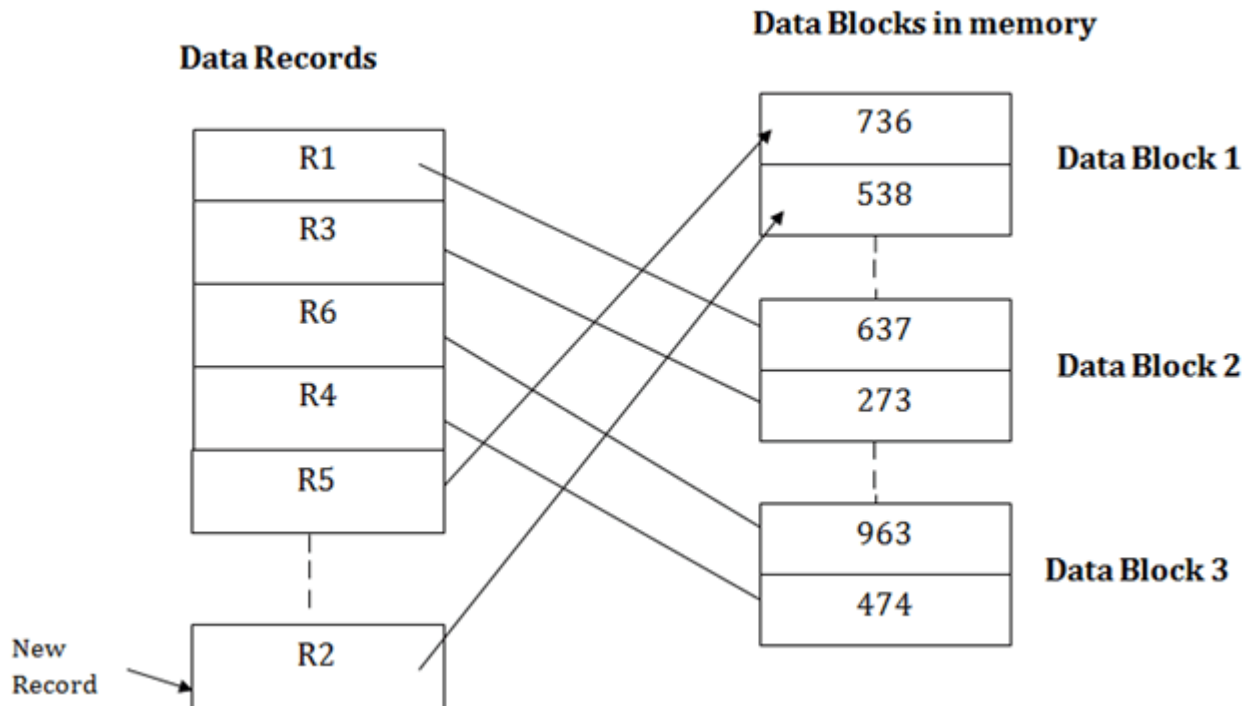
Heap file organization

- It is the simplest and most basic type of organization. It works with data blocks. In heap file organization, the records are inserted at the file's end. When the records are inserted, it doesn't require the sorting and ordering of records.
- When the data block is full, the new record is stored in some other block. This new data block need not to be the very next data block, but it can select any data block in the memory to store new records. The heap file is also known as an unordered file.
- In the file, every record has a unique id, and every page in a file is of the same size. It is the DBMS responsibility to store and manage the new records.



Insertion of a new record

Suppose we have five records R1, R3, R6, R4 and R5 in a heap and suppose we want to insert a new record R2 in a heap. If the data block 3 is full then it will be inserted in any of the database selected by the DBMS, let's say data block 1.



If we want to search, update or delete the data in heap file organization, then we need to traverse the data from starting of the file till we get the requested record.

If the database is very large then searching, updating or deleting of record will be time-consuming because there is no sorting or ordering of records. In the heap file organization, we need to check all the data until we get the requested record.

Pros of Heap file organization

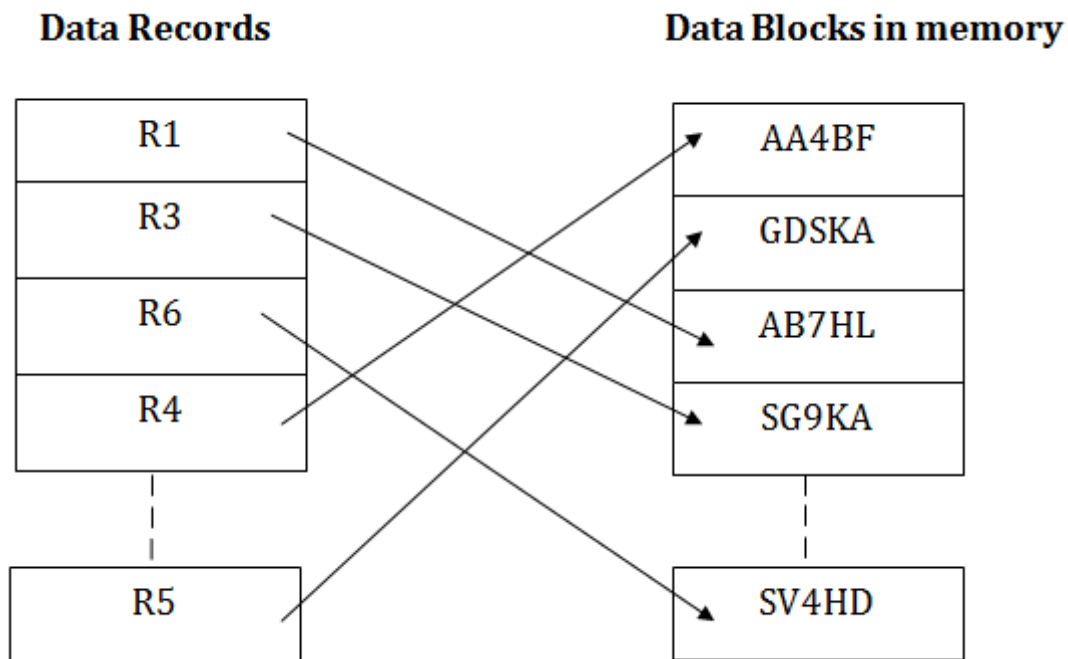
- It is a very good method of file organization for bulk insertion. If there is a large number of data which needs to load into the database at a time, then this method is best suited.
- In case of a small database, fetching and retrieving of records is faster than the sequential record.

Cons of Heap file organization

- This method is inefficient for the large database because it takes time to search or modify the record.
- This method is inefficient for large databases.

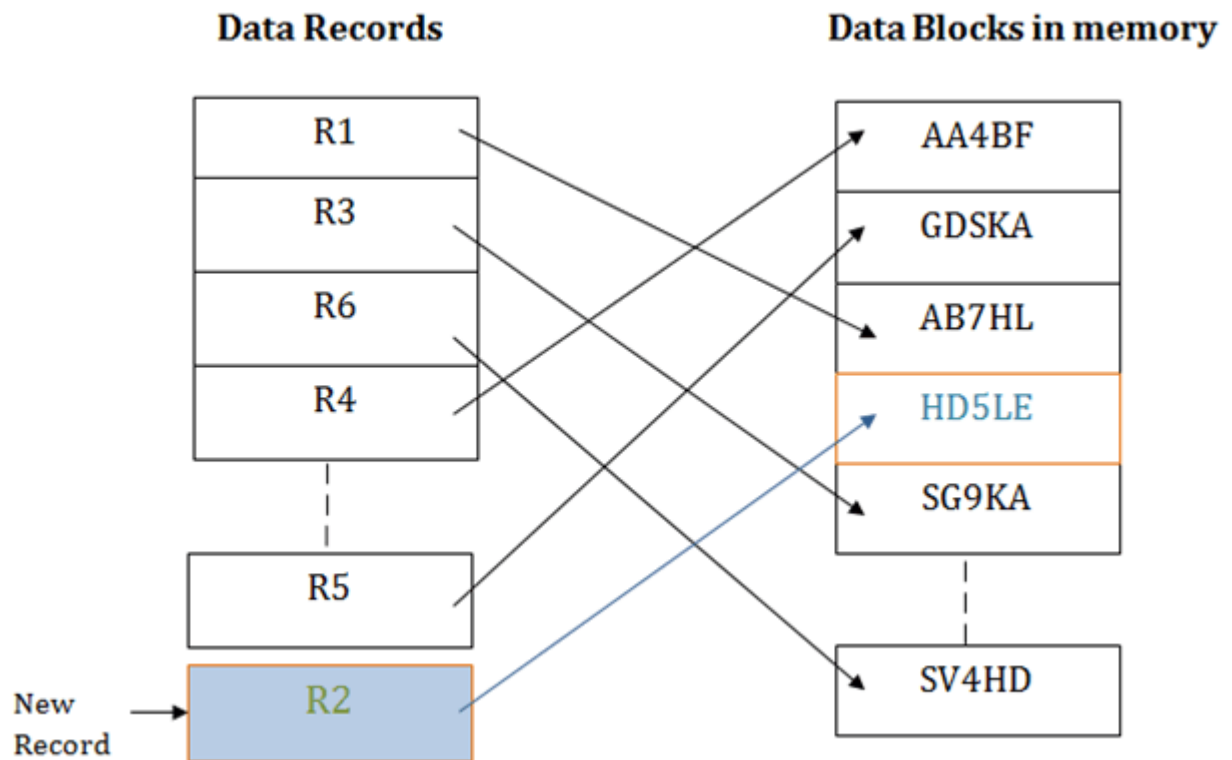
Hash File Organization

Hash File Organization uses the computation of hash function on some fields of the records. The hash function's output determines the location of disk block where the records are to be placed.



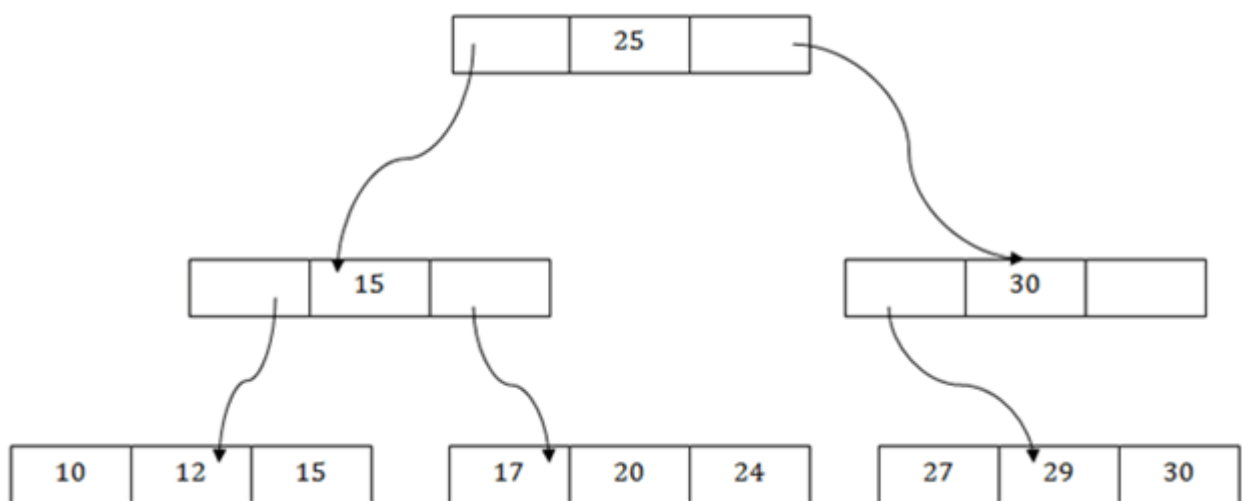
When a record has to be received using the hash key columns, then the address is generated, and the whole record is retrieved using that address. In the same way, when a new record has to be inserted, then the address is generated using the hash key and record is directly inserted. The same process is applied in the case of delete and update.

In this method, there is no effort for searching and sorting the entire file. In this method, each record will be stored randomly in the memory.



B+ File Organization

- B+ tree file organization is the advanced method of an indexed sequential access method. It uses a tree-like structure to store records in File.
- It uses the same concept of key-index where the primary key is used to sort the records. For each primary key, the value of the index is generated and mapped with the record.
- The B+ tree is similar to a binary search tree (BST), but it can have more than two children. In this method, all the records are stored only at the leaf node. Intermediate nodes act as a pointer to the leaf nodes. They do not contain any records.



The above B+ tree shows that:

- There is one root node of the tree, i.e., 25.
- There is an intermediary layer with nodes. They do not store the actual record. They have only pointers to the leaf node.
- The nodes to the left of the root node contain the prior value of the root and nodes to the right contain next value of the root, i.e., 15 and 30 respectively.
- There is only one leaf node which has only values, i.e., 10, 12, 17, 20, 24, 27 and 29.
- Searching for any record is easier as all the leaf nodes are balanced.
- In this method, searching any record can be traversed through the single path and accessed easily.

Pros of B+ tree file organization

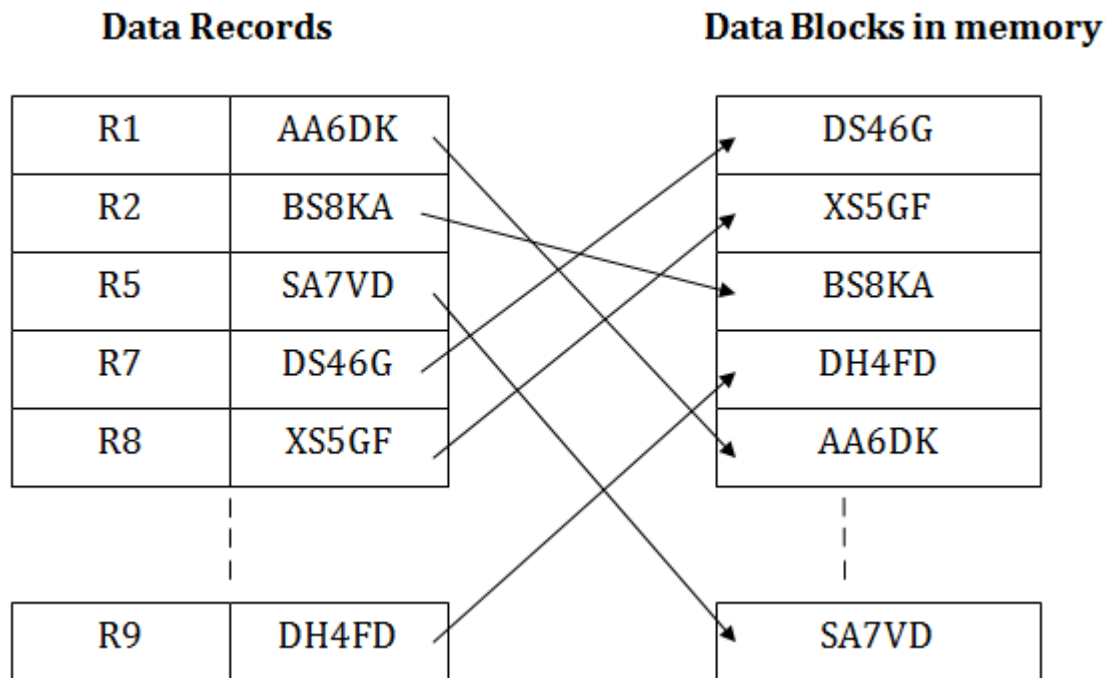
- In this method, searching becomes very easy as all the records are stored only in the leaf nodes and sorted the sequential linked list.
- Traversing through the tree structure is easier and faster.
- The size of the B+ tree has no restrictions, so the number of records can increase or decrease and the B+ tree structure can also grow or shrink.
- It is a balanced tree structure, and any insert/update/delete does not affect the performance of tree.

Cons of B+ tree file organization

- This method is inefficient for the static method.

Indexed sequential access method (ISAM)

ISAM method is an advanced sequential file organization. In this method, records are stored in the file using the primary key. An index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.



If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Pros of ISAM:

- In this method, each record has the address of its data block, searching a record in a huge database is quick and easy.
- This method supports range retrieval and partial retrieval of records. Since the index is based on the primary key values, we can retrieve the data for the given range of value. In the same way, the partial value can also be easily searched, i.e., the student name starting with 'JA' can be easily searched.

Cons of ISAM

- This method requires extra space in the disk to store the index value.
- When the new records are inserted, then these files have to be reconstructed to maintain the sequence.
- When the record is deleted, then the space used by it needs to be released. Otherwise, the performance of the database will slow down.

Cluster file organization

- When the two or more records are stored in the same file, it is known as clusters. These files will have two or more tables in the same data block, and key attributes which are used to map these tables together are stored only once.
- This method reduces the cost of searching for various records in different files.
- The cluster file organization is used when there is a frequent need for joining the tables with the same condition. These joins will give only a few records from both tables. In the given example, we are retrieving the record for only particular departments. This method can't be used to retrieve the record for the entire department.


EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key



DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

In this method, we can directly insert, update or delete any record. Data is sorted based on the key with which searching is done. Cluster key is a type of key with which joining of the table is performed.

Types of Cluster file organization:

Cluster file organization is of two types:

1. Indexed Clusters:

In indexed cluster, records are grouped based on the cluster key and stored together. The above EMPLOYEE and DEPARTMENT relationship is an example of an indexed cluster. Here, all the records are grouped based on the cluster key- DEP_ID and all the records are grouped.

2. Hash Clusters:

It is similar to the indexed cluster. In hash cluster, instead of storing the records based on the cluster key, we generate the value of the hash key for the cluster key and store the records with the same hash key value.

Keep Watching

Pros of Cluster file organization

- The cluster file organization is used when there is a frequent request for joining the tables with same joining condition.
- It provides the efficient result when there is a 1:M mapping between the tables.

Cons of Cluster file organization

- This method has the low performance for the very large database.
- If there is any change in joining condition, then this method cannot use. If we change the condition of joining then traversing the file takes a lot of time.
- This method is not suitable for a table with a 1:1 condition.

17B) Explain sparse and dense indices in detail with examples?

Ans: Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.

- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.

UP	•	→	UP	Agra	1,604,300
USA	•	→	USA	Chicago	2,789,378
Nepal	•	→	Nepal	Kathmandu	1,456,634
UK	•	→	UK	Cambridge	1,360,364

Sparse index

- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.

UP	•	→	UP	Agra	1,604,300
Nepal	•	→	USA	Chicago	2,789,378
UK	•	→	Nepal	Kathmandu	1,456,634
		→	UK	Cambridge	1,360,364

18. Find the major differences among primary, secondary and clustering indexes?

Ans:

PRIMARY INDEX VERSUS SECONDARY INDEX

PRIMARY INDEX

Index on a set of fields that includes the unique primary key and is guaranteed not to contain duplicates

Requires the rows in data blocks to be ordered on the index key

There is only one primary index

There are no duplicates in the primary key

SECONDARY INDEX

Index that is not a primary index and may have duplicates

Does not have an impact on how the rows are actually organized in data blocks

There can be multiple secondary indexes

There can be duplicates in the secondary index

Visit www.PEDIAA.COM

Clustering Index in DBMS

In a clustered index, records themselves are stored in the Index and not pointers. Sometimes the Index is created on non-primary key columns which might not be unique for each record. In such a situation, you can group two or more columns to get the unique values and create an index which is called clustered Index. This also helps you to identify the record faster.

Example:

Let's assume that a company recruited many employees in various departments. In this case, clustering indexing in DBMS should be created for all employees who belong to the same dept.

It is considered in a single cluster, and index points point to the cluster as a whole. Here, Department_no is a non-unique key.

19. Explain about B+ Tree and the structure of B+ Tree in detail with suitable examples.

Ans: B+ Tree

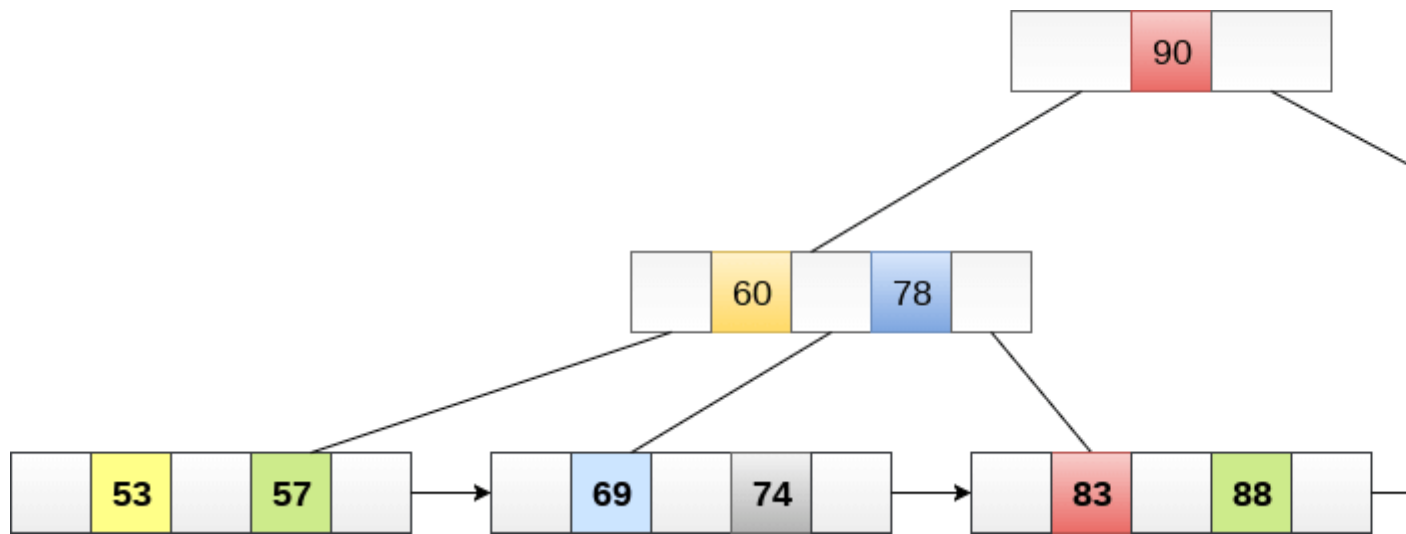
B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.

In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.

The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.

B+ Tree are used to store the large amount of data which can not be stored in the main memory. Due to the fact that, size of main memory is always limited, the internal nodes (keys to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.

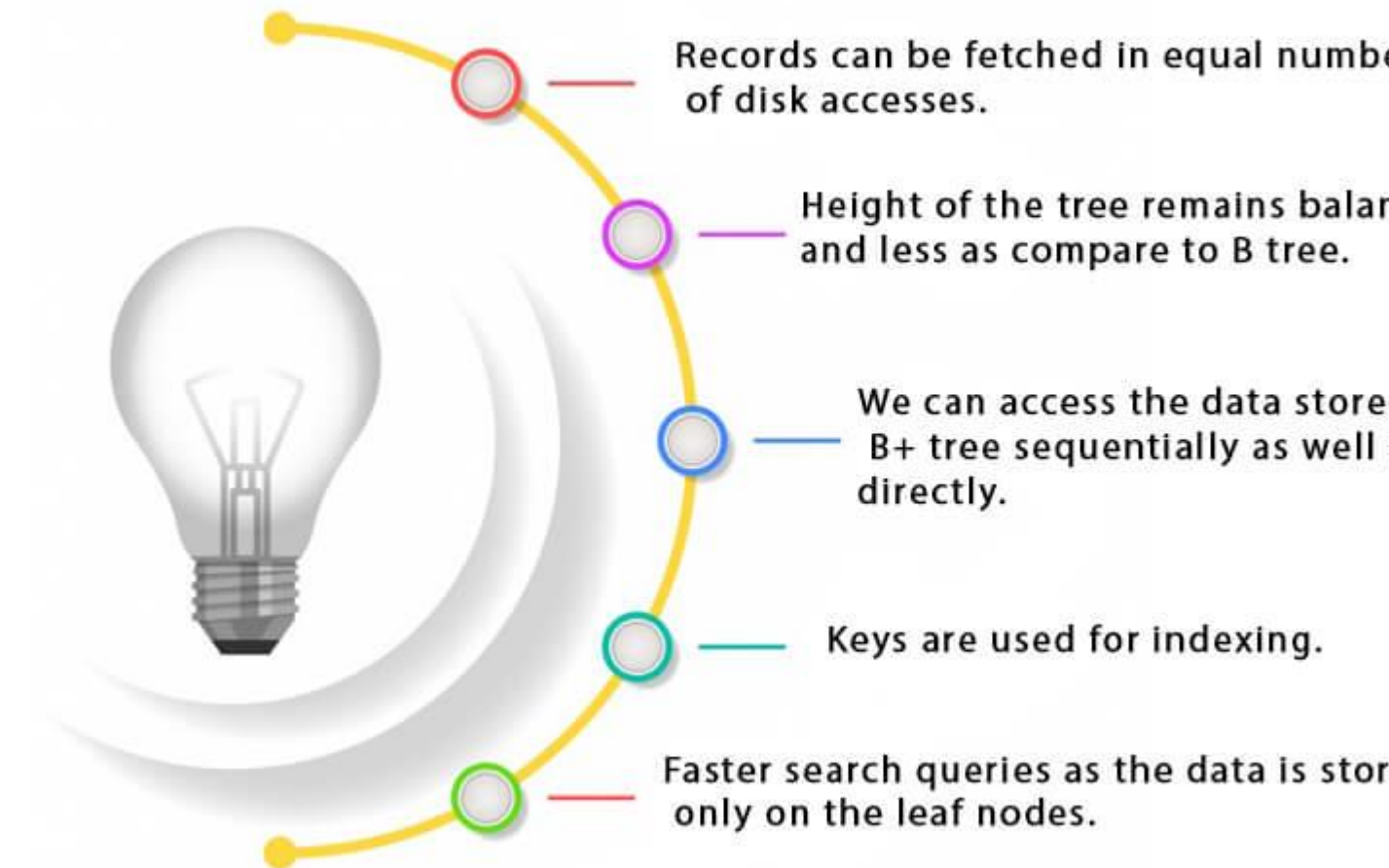
The internal nodes of B+ tree are often called index nodes. A B+ tree of order 3 is shown in the following figure.



Advantages of B+ Tree

1. Records can be fetched in equal number of disk accesses.
2. Height of the tree remains balanced and less as compare to B tree.
3. We can access the data stored in a B+ tree sequentially as well as directly.
4. Keys are used for indexing.
5. Faster search queries as the data is stored only on the leaf nodes.

Advantages of B+ Tree



Insertion in B+ Tree

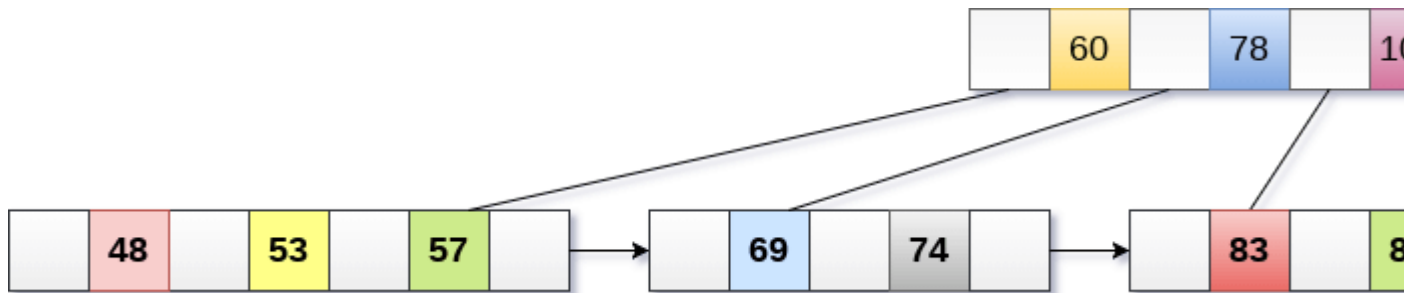
Step 1: Insert the new node as a leaf node

Step 2: If the leaf doesn't have required space, split the node and copy the middle node to the next index node.

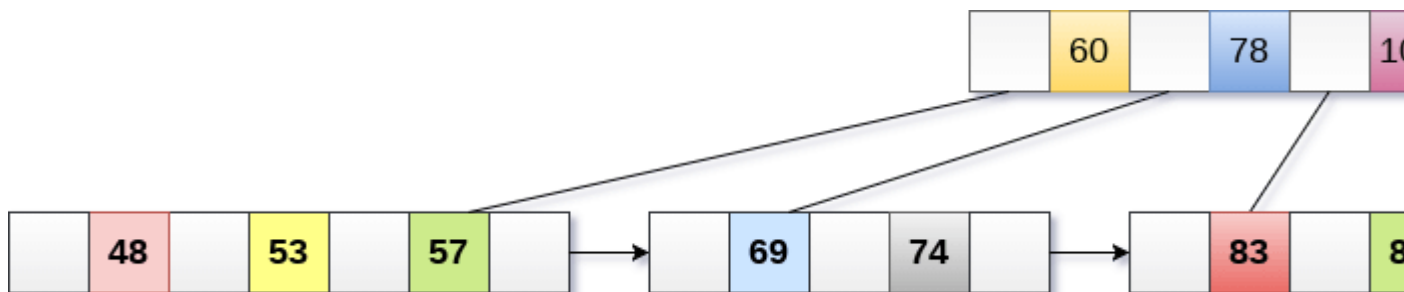
Step 3: If the index node doesn't have required space, split the node and copy the middle element to the next index page.

Example :

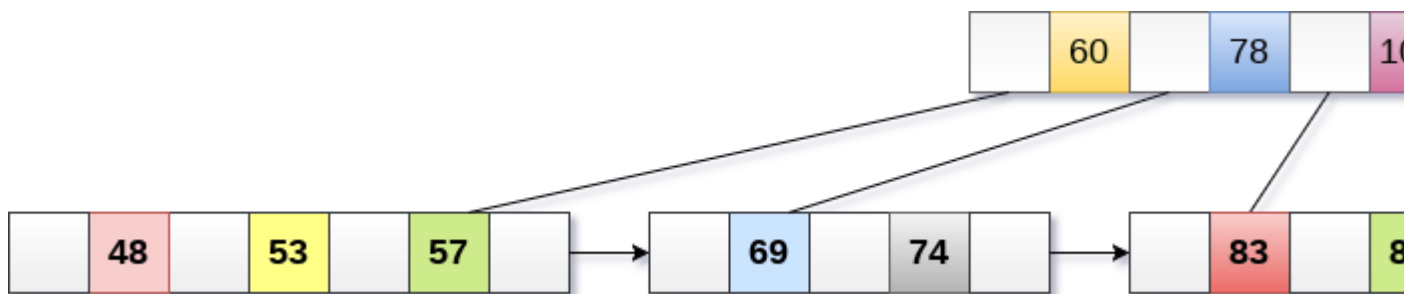
Insert the value 195 into the B+ tree of order 5 shown in the following figure.



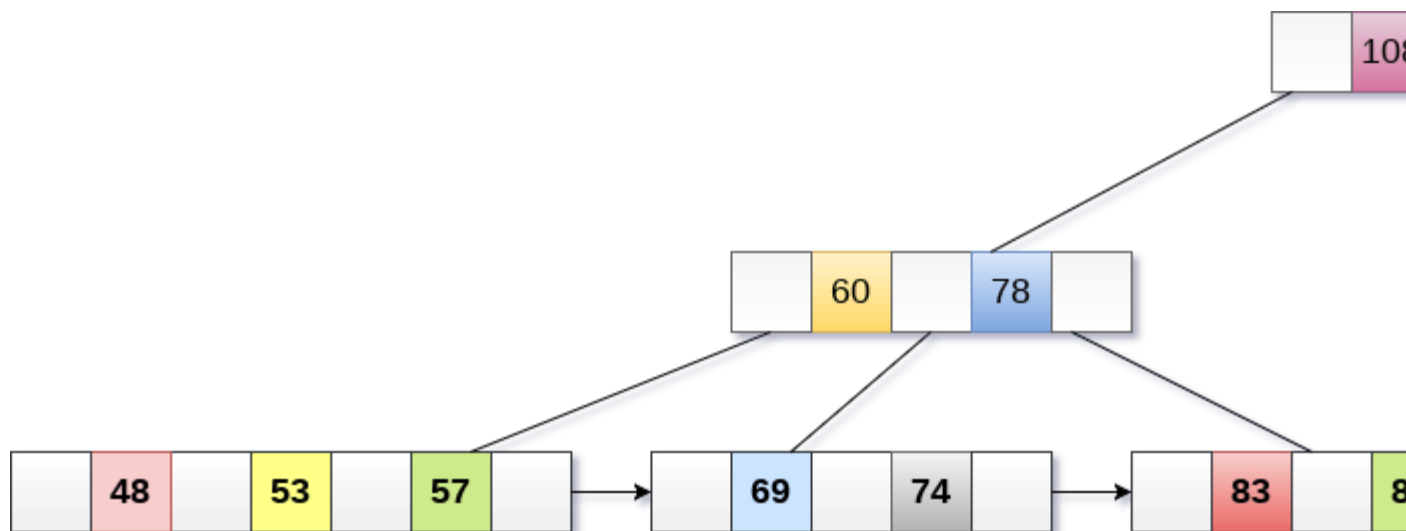
195 will be inserted in the right sub-tree of 120 after 190. Insert it at the desired position.



The node contains greater than the maximum number of elements i.e. 4, therefore split it and place the median node up to the parent.



Now, the index node contains 6 children and 5 keys which violates the B+ tree



Deletion in B+ Tree

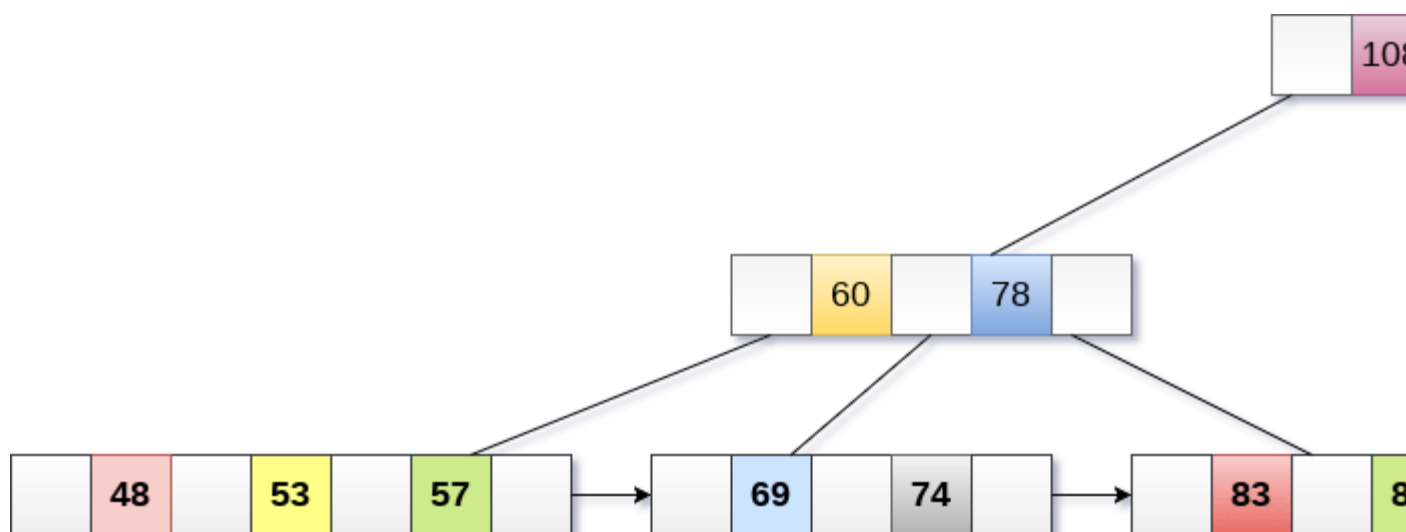
Step 1: Delete the key and data from the leaves.

Step 2: if the leaf node contains less than minimum number of elements, merge down the node with its sibling and delete the key in between them.

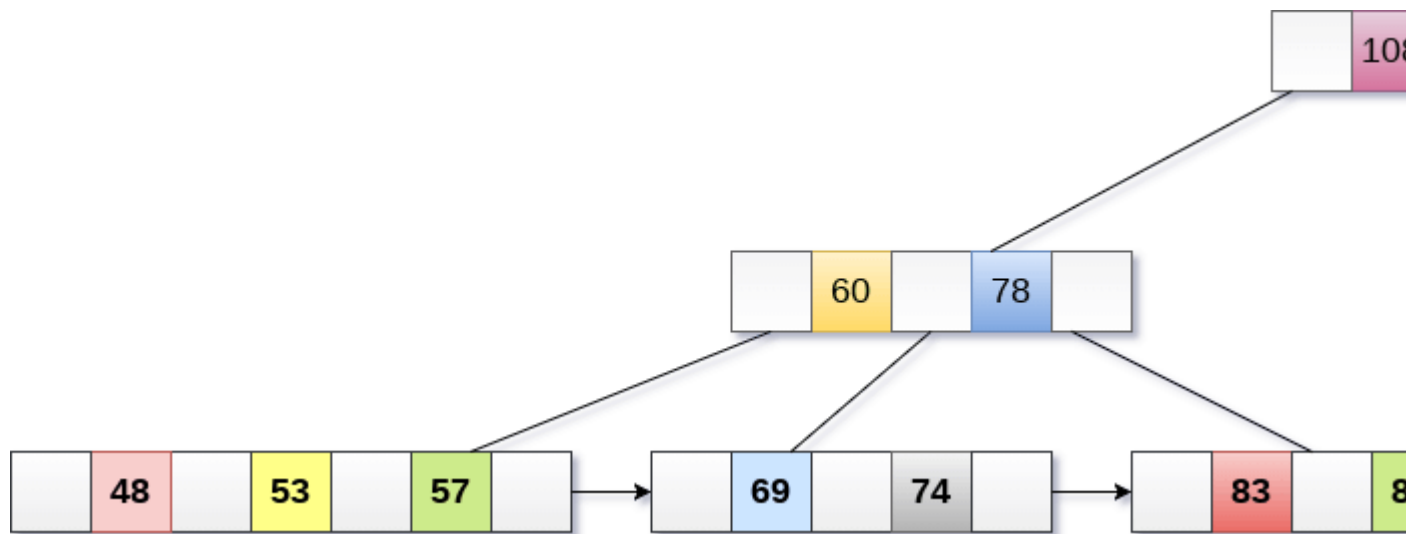
Step 3: if the index node contains less than minimum number of elements, merge the node with the sibling and move down the key in between them.

Example

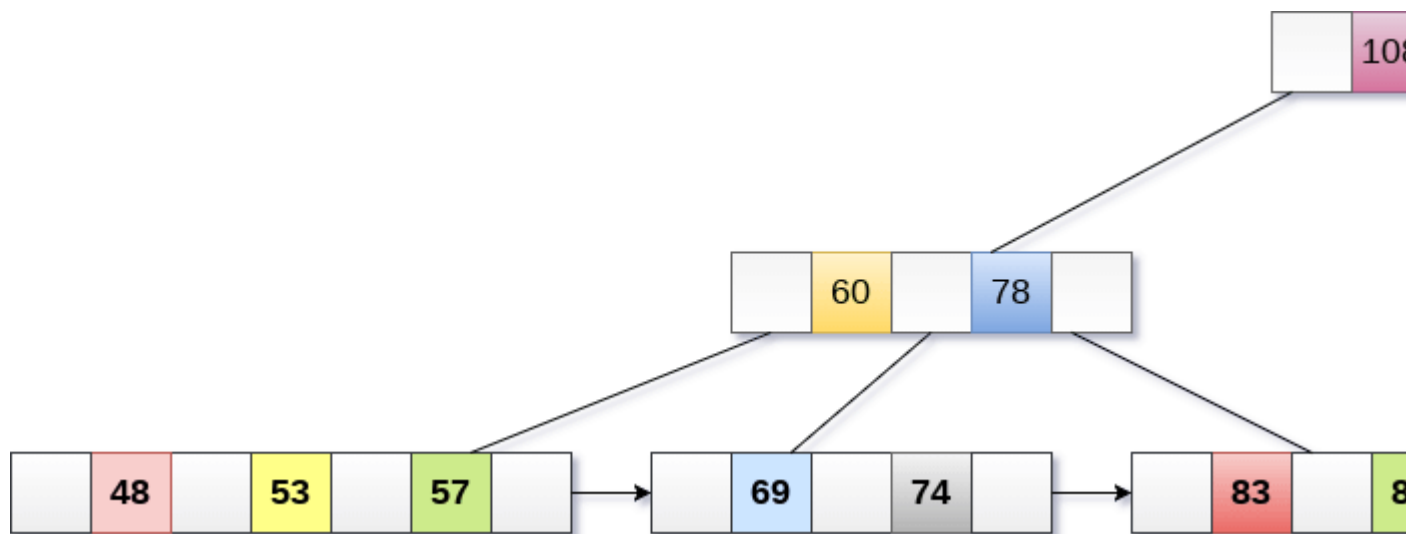
Delete the key 200 from the B+ Tree shown in the following figure.



200 is present in the right sub-tree of 190, after 195. delete it.

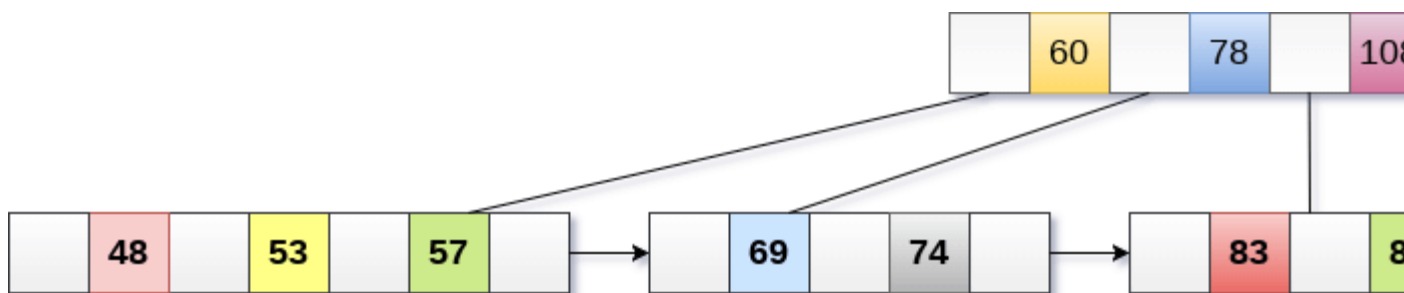


Merge the two nodes by using 195, 190, 154 and 129.



Now, element 120 is the single element present in the node which is violating the B+ Tree properties. Therefore, we need to merge it by using 60, 78, 108 and 120.

Now, the height of B+ tree will be decreased by 1.



20. Discussion detail about Linear Hashing with pros and cons.

Ans: **Hash** or [Hash table](#) is a data structure that maps keys to values using a special function called a **hash function**. Hash stores the data in an associative manner. It stores data in an array where each data value has its own unique index. Hashing provides secure access and retrieval of the data.

Collision Resolution: Collision resolution in hash can be done by two methods:

- [Open addressing](#) and
- Closed addressing.

Open Addressing: Open addressing collision resolution technique involves generating a location for storing or searching the data called **probe**. It can be done in the following ways:

- **Linear Probing:** If there is a collision at i then we use the hash function – $H(k, i) = [H'(k) + i] \% m$
where, i is the index, m is the size of hash table $H(k, i)$ and $H'(k)$ are hash functions.
- **Quadratic Probing:** If there is a collision at i then we use the hash function – $H(k, i) = [H'(k) + c_1 * i + c_2 * i^2] \% m$
where, i is the index, m is the size of hash table $H(k, i)$ and $H'(k)$ are hash functions, c_1 and c_2 are constants.
- **Double Hashing:** If there is a collision at i then we use the hash function – $H(k, i) = [H_1(k, i) + i * H_2(k)] \% m$
where, i is the index, m is the size of hash table $H(k, i)$, $H_1(k) = k \% m$ and $H_2(k) = k \% m'$ are hash functions.

Closed Addressing:

Closed addressing collision resolution technique involves chaining. Chaining in the hashing involves both array and linked list. In this method, we generate a probe with the help of the hash function and link the keys to the respective index one after the other in the same index. Hence, resolving the collision.

Applications of Hash:

- Hash is used in databases for indexing.
- Hash is used in disk based data structures.
- In some programming languages like Python, JavaScript hash is used to implement objects.

Real-Time Applications of Hash:

- Hash is used for cache mapping for fast access of the data.
- Hash can be used for password verification.
- Hash is used in cryptography as a message digest.

Advantages of Hash:

- Hash provides better synchronization than other data structures.
- Hash tables are more efficient than search trees or other data structures.
- Hash provides constant time for searching, insertion and deletion operations on average.

Disadvantages of Hash:

- Hash is inefficient when there are many collisions.
- Hash collisions are practically not be avoided for large set of possible keys.
- Hash does not allow null values.