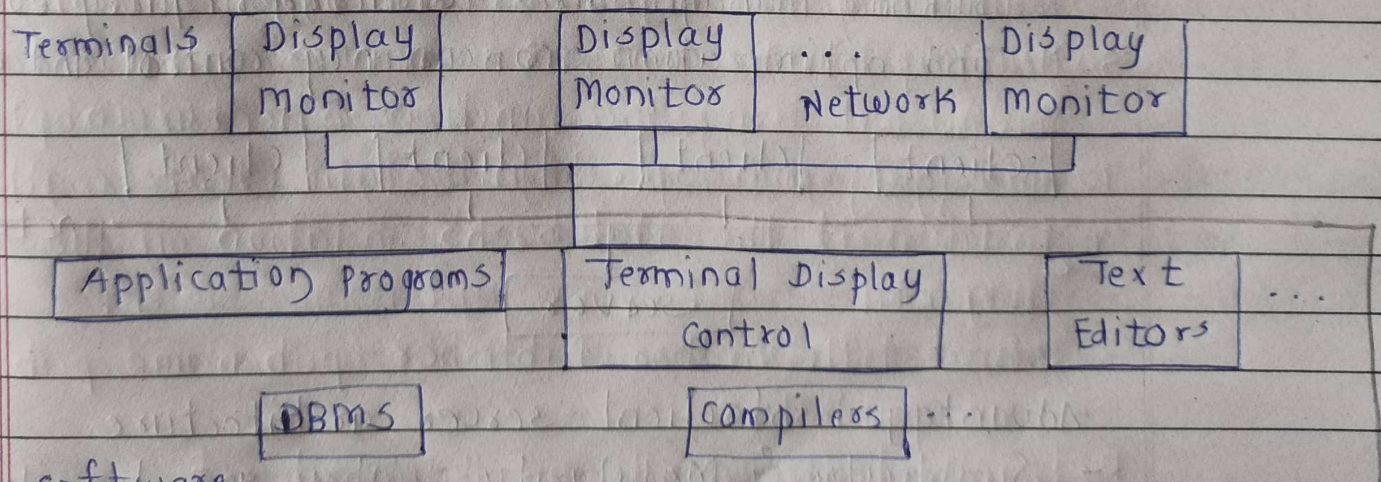
ISE 1 - Assignment

Date
Page

1. Describe centralized and client-server database architecture in detail.

Centralized database architecture

1. Centralized Database systems are those that run on a single computer system and do not interact with other computer systems to high performance database system running on high-end server system
2. In centralized architecture, the mainframe computers are used for processing all system's functions including User application programs and user interface programs as well as DBMS functionalities
3. This is because in earlier days, most users accessed such systems via Computer Terminals, which can't process and they have only display capability
4. 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
5. General purpose computer system is one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.



Client-Server Architecture

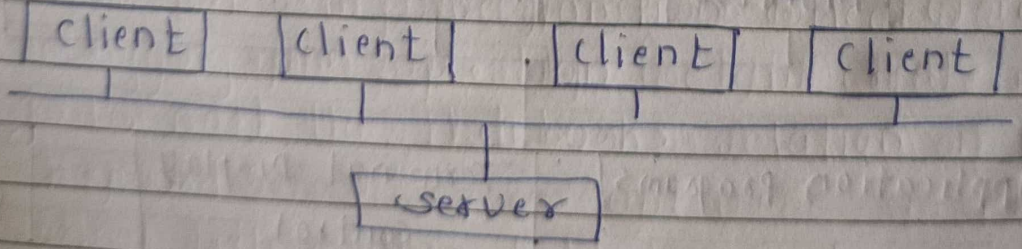
1. Client / server architecture is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client.
2. This type of architecture has one or more client computers connected to a central server over a network or internet connection.
3. This system shares computing resources. Client-server architecture of database system has two logical compo.

i) Client - clients are personal computers or work-stations. The applications and tools run on one or more client machines.

The client computer is called front-end. Front-end consists of tools such as forms, report-writers and graphical user interface facilities.

ii) Server - Servers are provided with specific functionalities. Server is large workstation, mini frame or mainframe computer system.

The server computer is called backend compute. Backend manages access structures, query evaluation and optimization, concurrency control and recovery.



Advantages of client-server architecture

- Simplicity & Modularity - They are loosely coupled
- Flexibility - Can be easily migrated across different machines
- Concurrency
- Better functionality in terms of costs.

2. Explain array and multiset collection types supported by SQL and construct queries for the songs table to be created using song structured type containing song-title varchar, composer array, release-date of date type, music company of company type (company_name, company_branch) and a lyrics of array.

Array - An array is a collection of values of the same data type, which can be stored and accessed as a single entity.

SQL DBMS provides support for arrays, allowing us to store and query data in more structured manner.

To create an array type, we use the CREATE TYPE statement in SQL. Here's an example:

```
CREATE TYPE colors_array AS VARRAY(5) OF VARCHAR(20)
```

In the above code, we create a new array type called colors_array, which can store up to five strings of maximum length 20 characters.

Multiset - Multiset is a collection of values where duplicate values are allowed. Unlike arrays, multisets don't have a fixed length and can grow dynamically.

To create a multiset type, we use the CREATE TYPE statement, along with the MULTISET keyword. Here's an example:

```
CREATE TYPE fruits_multiset AS TABLE OF VARCHAR(20);
```

In the above code, we create a new multiset type called fruits_multiset, which can store an arbitrary number of strings of maximum length 20 characters.

The queries for the given is as below:

```
CREATE TYPE company_type AS (
    company_name VARCHAR(100),
    company_branch VARCHAR(100),
);
```

```
CREATE TYPE song_type AS (
    song_title VARCHAR(20),
    composer VARCHAR(200) ARRAY[10],
    release_date DATE,
    music_company company_type,
    lyrics VARCHAR(1000) ARRAY[100]
);

CREATE TABLE songs (
    song_id int PRIMARY KEY,
    song_info song_type
);
```

3. Explain Nesting and Un-Nesting with one example.

Nesting

1. Nesting refers to organizing data in a hierarchical structure, where one or more attributes contain collections or sub-tables (arrays, multiset, or structured types) within a single column of a table.
2. Nesting simplifies data modeling by allowing related attributes to be grouped together, which reduce redundancy and improves data organization.
3. Nesting can be done in a manner similar to aggregation, but using the function collect() in place of an aggregation operation, we can create multiset.

e.g. To nest the flat_books relation on the attribute keyword:

```
select title, author, Publisher (pub_name, pub_branch)
    as publisher,
    collect (keyword) as keyword_set
from flat_books
group by title, author, publisher
```

e.g. To nest on both ... and keywords (

```
select title, collect(author) as author_set,
    Publisher(pub_name, pub_branch) as publisher,
    collect(keyword) as keyword_set
from flat_books
group by title, publisher
```

Unnesting

1. The transformation of a nested relation into a form with fewer (or no) relation-valued attributes is called unnesting
2. Unnesting transforms the hierarchical data back into a flat structure. It is useful for querying or displaying the data in a non-hierarchical manner.
3. It is also useful for reporting and analysis.
4. We can retrieve the nested data as individual rows using SQL's UNNEST() function

e.g.

```
SELECT student_id, student_name, cg.course_name, cg.grade
FROM students, UNNEST(course_grades) AS cg(course_name, grade);
```

This query breaks down the array of course_grades into individual rows for each student.

The result would look something like this:

student_id	student_name	course_name	grade
1	John Doe	Math	A
1	John Doe	Science	B
1	John Doe	History	A

4. Explain object identity and reference types in SQL.

Object Types and User-Defined Types

1. Object-relational DBMS allows us to define types called object types similar to the data types of SQL.
2. That means that object type can be used like any other built-in datatype.

3. For example, we define an account type for the issue-tracking database as follows:

```
CREATE OR REPLACE TYPE account_type AS OBJECT (
    account_id    INTEGER,
    account_name  VARCHAR(20),
    email         VARCHAR(100));
```

4. "AS OBJECT" is used to create an object type.
5. The use of "OR REPLACE" is syntactically optional.
6. Now, we can create a table containing only objects of the account type as follows.

```
CREATE TABLE accounts OF account_type (
    PRIMARY KEY (account_id),
    UNIQUE (account_name));
```

7. The table accounts is a single-column table & is called an object table.
8. Each of its rows represents an object of the type account_type. We can populate it just like populating a usual table as follows.

```
INSERT INTO accounts (account_id, account_name, email)
VALUES (2, 'alice', 'alice@abc.com');
```

9. The other way is to use the built-in constructor method of the type account_type as follows.

```
INSERT INTO accounts
VALUES (account_type(1, 'bob', 'bon@abc.com'));
```

10. We can query an object table like a pure relational table as follows.

```
SELECT account_id, account_name, email
FROM accounts;
```

Reference Type

1. For every user-defined type T, there is REF T as the type of references to instances (values) of the type T.

2. The reference type is used to model one-to-one, one-to-many and many-to-many relationships.
3. e.g. the attributes reported_by and assigned_to in the issue type defined in the following statement are of the type of references to values of the account type.

```
CREATE TYPE issue_type AS OBJECT (
    issue_id    INTEGER,
    summary     VARCHAR(50),
    priority    VARCHAR(20),
    reported_by REF account_type,
    assigned_to REF account_type) NOT FINAL;
```

4. We call reported_by and assigned_to REF variables.
5. Two REF variables can be compared if, and only if, the values that they refer to are both of the same type, or one is a subtype of the other.
6. REF variables can only be compared for equality.
7. Declaring the issue type as "NOT FINAL" allows us to extend it.

5. Explain type inheritance and table inheritance with one example.

Inheritance enables you to share attributes between objects such that a subclass inherits attributes from its parent class.

1) Type Inheritance

- Type inheritance applies to named row types only.
- You can use inheritance to a group named row types into a type hierarchy in which each subtype inherits the representation and the behaviour of the supertype under which it is defined.
- It ensures consistent reuse of schema components
- It ensures that no data fields are accidentally left

- Suppose that we have following type definition for people.

```
create type Person
    (name varchar(20),
     address varchar(20))
```

- Using inheritance to define the student and teacher types

```
create type Student under Person
    (degree varchar(20),
     department varchar(20))
create type Teacher under Person
    (salary integer,
     department varchar(20))
```

- Subtypes can redefine methods by using overriding meth
- Multiple Type Inheritance:

```
create type Teaching Assistant
    under Student, Teacher
```

To avoid conflict between the two occurrences of department we can rename them.

```
create type Teaching Assistant
    under Student with (department as student-dept),
          Teacher with (department as teacher-dept)
```

2) Table Inheritance

- Only tables that are defined on named row types support table inheritance.
- Table inheritance is the property that allows a table to inherit the behaviour from the supertable above it in the table hierarchy.
- Tables created from subtypes can further be specified as subtables.

```
e.g. create table people of Person;
     create table students of Student under people;
     create table teachers of Teacher under people;
```

- Tuples added to a subtable are automatically visible to queries on the supertable.

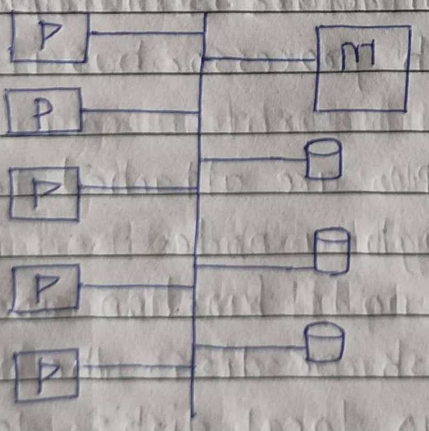
e.g. Query on people ... teachers.

- multiple table inheritance is possible, but is not supported in sql currently
e.g. teaching_assistants under students and teachers

6. Write short note on:

a) shared memory Architecture

- In shared memory architecture, there are multiple CPUs that are attached to an interconnection network.
- They are able to share a single or global main memory and common disk arrays.
- In this architecture, a single copy of a multi-threaded operating system and multithreded DBMS can support these multiple CPUs.
- Processors and disks have access to a common memory, typically via a bus or through an interconnection network.
- Architecture is not scalable beyond 32 or 64 processors since the bus or the interconnection network becomes a bottleneck.
- It is widely used for lower degrees of parallelism

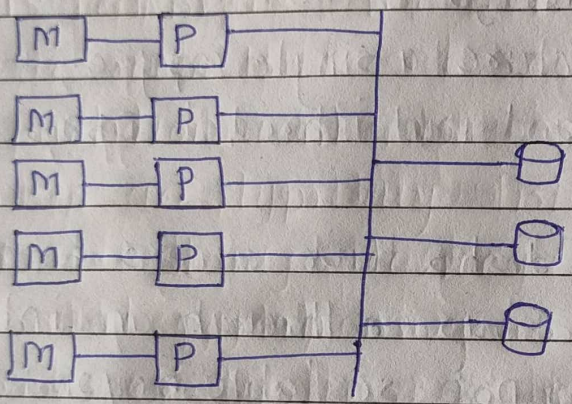


shared memory architecture

b) shared disk Architecture

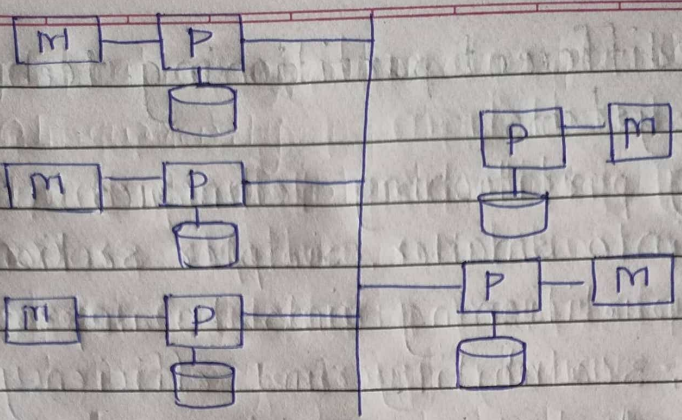
- In shared disk architecture, all processors can directly access all disks via an interconnection network, but the processors have private memories.
- The memory bus is not a bottleneck.

- If a processor fails, the other processors can take over its tasks since the database is resident on disks that are accessible from all processors.
- This architecture provides fault-tolerance.
- Bottleneck now occurs at interconnection to the disk subsystem.
- shared disk systems can scale to a somewhat larger number of processors, but communication between processors is slower.



shared disk architecture

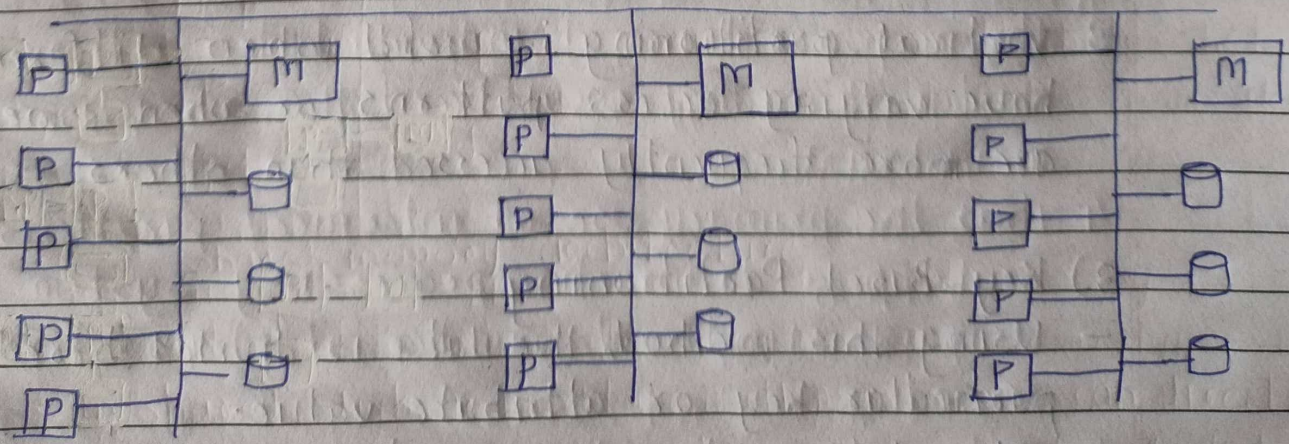
c) Shared Nothing Architecture
- Shared nothing architecture is multiple processor architecture in which each processor has its own memory & disk storage.
- In this, multiple CPUs are attached to an interconnection network through a node.
- ~~Each node of~~ A node function as the server for the data on the disk or disks the node owns.
- Data accessed from local disks do not pass through interconnection network, thereby minimizing the interference of resource sharing.
- shared nothing multiprocessors can be scaled up to thousands of processors without interference.
- Cost of communication and non-local disk access, sending data involves software interaction at both ends.



shared nothing architecture

d) Hierarchical Architecture

- It combines the characteristics of shared memory, shared disk and shared nothing architecture.
- In this, nodes connected by an interconnection network, and do not share disks or memory with each other.
- Each node of the system could be a shared memory system with a few processors.
- Alternatively, each node could be a shared disk system and each of the systems sharing a set of disks could be a shared-memory system.
- It is scalable due to availability of more memory and many processor.
- It is costly to other architecture.



Hierarchical Architecture

7. Illustrate different partitioning techniques in parallel database systems.

Using partitioning techniques, a huge dataset can be divided into smaller, simpler sections.

Data partitioning aims to improve data processing performance, scalability and efficiency. There are different partitioning techniques, which are as follows:

1) Horizontal Partitioning
- In this technique, the dataset is divided based on rows or records.
- Each partition contains a subset of rows, and the partitions are typically distributed across multiple servers or storage devices.
- Horizontal partitioning is often used in distributed databases or systems to improve parallelism and enable load balancing.

2) Vertical Partitioning
- Vertical partitioning divides the dataset based on columns or attributes.
- In this technique, each partition contains a subset of columns for each row.
- Vertical partitioning is useful when different columns have varying access patterns or when some columns are more frequently accessed than others.

3) Key-Based Partitioning
- Using this method, the data is divided based on a particular key or attribute value.
- The dataset has been partitioned, with each containing all the data related to a specific key value.

4) Range Partitioning
- Range partitioning divides the dataset according to a predetermined range of values.
- We can divide data based on a particular time range, for instance, if our dataset contains timestamps.
- When we want to distribute data evenly based on the range of values and have data with natural ordering, range partitioning can be helpful.

5) Hash-Based Partitioning
- Hash partitioning is the process of analyzing the data using a hash function to decide which division it belongs to.
- The data is fed into the hash function, which produces a hash value used to categorize the data into a certain divisi
- By randomly distributing data among partitions, hash-based partitioning can help with load balancing and quick data retrieval.

6) Round-robin Partitioning
- In round-robin partitioning, data is evenly distributed across partitions in a cyclic manner.
- Each partition is assigned the next available data item sequentially, regardless of the data's characteristics.
- Round-robin partitioning is straightforward to implement and can provide a basic level of load balancing.
- It is best suited for sequentially scanning the entire relation for each query.
- Here Range queries are difficult to process.