CHAPTER 5

Other Databases

Contents...

Objectives...

- To learn Parallel Databases
- To study Distributed Databases
- To understand XML and NoSQL Databases
- To study Concepts of Multimedia Databases
- To learn Big Data Databases

5.0 INTRODUCTION

- In the year 1970's big meant megabytes, subsequently with the increasing data needs, it grew to gigabytes and terabytes and further to yottabyte with the increase in digital information.
- The traditional world of relational database systems like Oracle RDBMS etc., faced challenges in storing large quantities/amount of data and needed to scale databases to data volumes beyond the storage and/or processing capabilities of a single large computer system.
- Many efforts have been made to store and manage data being generated from everywhere on the web (Internet). Several database management systems were proposed on the basis or master/slave, cluster computing or partitioning architecture like IBM DB2 partitioning, VoltTB etc.
- However, the problems in reliance on shared facilities and resources (CPU, Disk, and Processors), scalability and complex administration limitations, augmented by lack of support for critical requirements, led to development of 'shared nothing' architectures in the year 1980's.
- These systems focused on parallel and distributed data computation and solved big data problems using parallel computations. By the year 1990's, even these solutions faced challenges in running OLTP (OnLine Transactional Processing) and queries due to data overload.
- To provide solutions to these problems, Google responded with its GFS (Google File System is a scalable Distributed File System (DFS) created by Google Inc. and developed to accommodate Google's

expanding data processing requirements) followed by a powerful programming paradigm of MapReduce.

- Thereafter, a spectrum of new technologies emerged as the NoSQL movement stating a broad class of database management system to support increasing data storage and analytical requirements.
- Various NoSQL data stores like Cassandra, MongoDB and Hadoop HBASE etc., are in use today to acquire, manage, store and query big data.
- NoSQL databases are inherently schema-less and permit records to have variable number of fields, making them distinct from other non-relational databases like hierarchical databases and object-oriented databases.
- These are highly scalable and well suited for dynamic data structures. NoSQL data is characterized by being basically available and eventually consistent.

5.1 INTRODUCTION TO PARALLEL AND DISTRIBUTED DATABASES

5.1.1 Parallel Databases

- A parallel processing involves multiple processes that are active simultaneously and solving a given problem, generally on multiple processors (or nodes).
- Parallel processing exploits multiprocessor computers to run application programs by using several processors cooperatively, in order to improve performance.
- Parallel processing divides a large task into many smaller tasks and executes the smaller tasks concurrently on several CPU's and completes it more quickly.
- A parallel database is designed to take advantage of executing operations in parallel, by running multiple instances that share a single physical database.
- A parallel server processes transactions in parallel by servicing a stream of transactions using multiple processors on different sites, where each processor processes an entire transaction.
- To improve system performance, a parallel database system allows multiple users to access a single physical database from multiple machines.
- To balance the workload among processors, parallel databases provide concurrent access to data and preserve data integrity.
- Parallel database system is a DBMS running across multiple processors and disks that is designed to perform various tasks concurrently like loading data, building indexes and evaluating queries.
- A parallel DBMS can be defined as, "a DBMS implemented on a tightly coupled multiprocessor system". In tightly coupled systems, there is a single system-wide global primary memory (address space) that is shared by all processors connected to the system.
- Fig. 5.1 shows typical parallel database system.

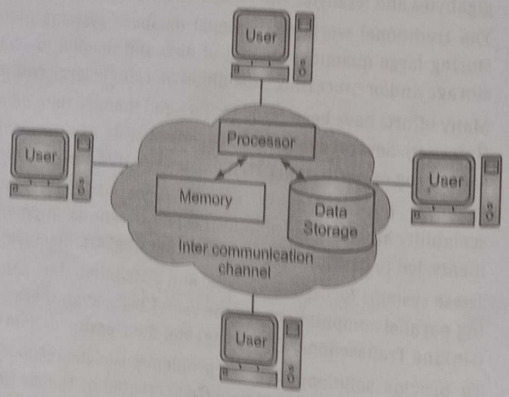


Fig. 5.1: Typical Parallel Database System

- There are several architectures (model) can be used for parallel database systems, like Shared Memory Architecture, Shared Disk Architecture, Shared Nothing Architecture, Hierarchical Architecture.

1. **Shared Memory Architecture:** In shared memory architecture, there are many (multiple) CPUs that are attached to an interconnection network. They can share a single or global main memory and common disk arrays (storages)
2. **Shared Disk Architecture:** In shared disk architecture, multiple CPUs are attached to an interconnection network. Each CPU has its own memory but all have access to the same disk.
3. **Shared Nothing Architecture:** In shared nothing architecture no two CPUs can access the same disk area. There is no sharing of memory or disk resources.
4. **Hierarchical Architecture:** Hierarchical architecture is a hybrid (mixed/combined) of shared memory, shared disk and shared nothing architectures. Hierarchical architecture is also known as Non-Uniform Memory Architecture (NUMA).

Advantages of Parallel Databases:

1. **High Availability:** Data availability in a parallel database system is much higher than that in a centralized DBMS in case of failure. In parallel database system, the data can be copied to multiple locations to improve the availability of data.
2. **Better Performance/High Processing Speed:** In parallel database system, for processing an application higher speed-up (the amount of time needed to complete a single task) and scale-up (more number of tasks can be completed in a given time interval) can be attained by involving a number of processors. The performance of the system can be improved by connecting multiple CPU and disks in parallel. These databases allowing faster access to very large databases.
3. **Reliability:** A parallel database, properly configured, can continue to work despite the failure of any computer in the cluster. The database server senses that a specific computer is not responding and reroutes its work to the remaining computers.
4. **Greater Flexibility:** Parallel server environments are extremely flexible.
5. **Serves Multiple Users:** In parallel database system, a single system can be serve thousands of users.

Disadvantages of Parallel Databases:

1. The startup costs of parallel databases are comparatively high.
2. Existing CPU's get slow down, as more CPU's are added.
3. Number of resources required is large thus cost is increased.
4. Also due to large number of resources complexity is increased.

5.1.2 Distributed Databases

- In the 1980s, distributed database systems had evolved to overcome the limitations of centralized database management systems and to cope up with the rapid changes in communication and database technologies.
- A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.
- Distributed database can be defined as, "logically interrelated collection of shared data physically distributed over a computer network".

OR

- We can also define a Distributed DataBase (DDB) as, "a collection of multiple logically interrelated databases distributed over a computer network or Internet".

OR

- A distributed database is defined as, "a logically interrelated collection of shared data, and a description of this data is physically distributed over a computer network".
- The software system that permits the management of the distributed database and makes the distribution transparent to the users is known as a Distributed DBMS.
- In simple words, a logically interrelated collection of shared data physically distributed over a computer network is called as a distributed database. And the software system that permits the management of the distributed database and makes it transparent to the users is called as a Distributed DBMS (DDBMS).
- A distributed database is a database that is under the control of a central DBMS in which not all storage devices are attached to a common CPU. It may be stored on multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.

Features of Distributed Databases:

1. Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.
2. The processors in the sites are connected via a network. They do not have any multiprocessor configuration.
3. A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Distributed Database Management System (DDBMS):

- A DDBMS manages the distributed database and provides mechanisms so as to make the databases transparent to the users.
- A DDBMS is the software that manages the DDB and provides an access mechanism that makes this distribution transparent to the users.
- In these systems, data is intentionally distributed among multiple nodes so that all computing resources of the organization can be optimally used.
- DDBMS can be defined as, "a centralized software system that manages a distributed database while making the distribution transparent to the users".

Features of DDBMS:

1. DDBMS is used to create, retrieve, update and delete distributed databases.
2. It is designed for heterogeneous database platforms.
3. DDBMS is used in application areas where large volumes of data are processed and accessed by numerous users simultaneously.
4. DDBMS maintains confidentiality and data integrity of the databases.
5. It synchronizes the database periodically and provides access mechanisms by the virtue of which the distribution becomes transparent to the users.
6. DDBMS ensures that the data modified at any site is universally updated.

- A distributed database system allows applications to access various data items from local and remote databases.
- Applications in DDBMS are classified into two categories depending on whether the transactions access data from local site or remote site as given below:

1. **Local applications** require access to local data only and do not require data from more than one site.
2. **Global applications** require access to data from other remote sites in the distributed system.

- In distributed database system, the database is shared on several computers. The computers in a distributed system communicate with one another through various communication media, such as high-speed networks or telephone lines. They do not share main memory or disks.
- The computers in distributed system may vary in size and function, ranging from workstations up to mainframe systems.
- The computers in distributed system are referred to by a number of different names, such as sites or nodes depending on the context in which they are mentioned.
- The general structure of distributed system is shown in Fig. 5.2.

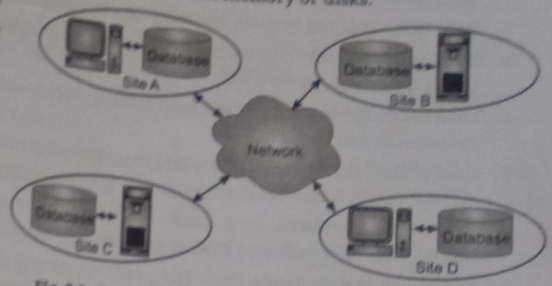


Fig. 5.2: General Structure of Distributed Database System

- DDBMS are of two types as explained below:

1. **Homogeneous DDBMS:** In a distributed system, if all sites use the same DBMS product, it is called a homogenous distributed database system.
2. **Heterogeneous DDBMS:** A distributed database has to be constructed by linking multiple already-existing database systems together, each with its own schema and possibly running different database management software such systems are called heterogeneous distributed database systems.

Advantages of DDBMS:

1. **Increased Efficiency:** DDBMS increased efficiency of processing by keeping the data close to the point where it is most frequently used.
2. **Sharing Data:** There is a provision in the environment where user at one site may be able to access the data residing at other sites.
3. **Improved Performance:** A DDBMS can provide improved performance since local data is maintained locally.
4. **Increased Accessibility:** DDBMS increased accessibility by allowing to access data between several sites (for example, say accessing an account of New Delhi from seating at New York and vice versa) via communication network.
5. **Increased Local Autonomy:** Where each site is able to retain degree of control over data that are stored locally.
6. **Improved Reliability/Availability:** When a database is distributed over many sites, most sites will continue to operate even if one or two of them fail.
7. **Easier Expansion:** Distributed systems are more modular, hence, they can be expanded easily as compared to centralized system.
8. **Integration of Existing Databases:** When several databases already exist in an organization and the necessity of performing global applications arises, distributed database is the natural solution.
9. **Speeding-up of Query Processing:** A distributed database system makes it possible to process data at several sites simultaneously. If a query involves data stored at several sites, it may be possible to split the query into a number of subqueries that can be executed in parallel. Thus, query processing becomes faster/quicker in distributed systems.

Disadvantages of DDBMS:

1. **Need for Complex and Expensive Software:** DDBMS demands complex and often expensive software to provide data transparency and co-ordination across the several sites.
2. **Data Integrity:** The need for updating data in multiple sites poses problems of data integrity.
3. **Lack of Standards:** There are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.
4. **Overheads for Improper Data Distribution:** Improper data distribution often leads to very slow response to user requests.
5. **Cost of Development:** The increased complexity and more extensive infrastructure of distributed systems require additional manpower for the design effort and additional costs.
6. **Processing Overhead:** Even simple operations may require a large number of communications and additional calculations to provide uniformity in data across the sites.
7. **Increased Maintenance Cost:** The procurement and maintenance costs of a distributed DBMS are much higher than those of a centralized system, as complexity increases.
8. **Security:** As data in a distributed DBMS are located at multiple sites, the probability of security lapses increases.

Difference between Parallel Database and Distributed Database:

Sr. No.	Characteristics	Parallel Database	Distributed Database
1.	Nature	Parallel databases are generally homogeneous in nature.	Distributed databases may be homogeneous or heterogeneous.
2.	Definition	It is software where multiple processors are used to execute and run queries in parallel.	It is software that manages multiple logically interrelated databases distributed over a computer network.
3.	Geographical location	The nodes are located at geographically same location.	The nodes are usually located at geographically different locations.
4.	Execution speed	Quicker/faster.	Slower.
5.	Overhead	Less.	More.
6.	Performance	Lower reliability and availability.	Higher reliability and availability.
7.	Scope of expansion	Difficult to expand.	Easier to expand.
8.	Backup	Backup at one site only.	Backup at multiple sites.
9.	Consistency	Maintaining consistency is easier.	Maintaining consistency is difficult.
10.	No. of Locations	Processors are tightly coupled and constitute a single database system i.e., parallel databases are centralized databases and data reside a single location.	Sites are loosely coupled and share no physical components i.e., distributed databases are geographically separated and data are distributed at several locations or sites.

Contd...

11.	Query optimization	More complex.	Query optimization techniques may be different at different sites and are easy to maintain.
12.	Size of Database	Very large.	Relatively small.

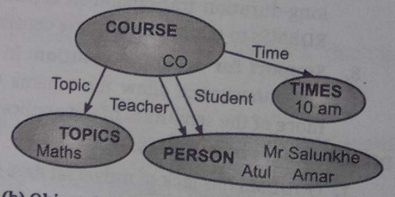
5.2 INTRODUCTION TO OBJECT BASED DATABASES

- Object-Oriented DataBases (OODBs) developed first as an approach to add persistence seamlessly into Object-Oriented Programming Languages (OOPLs).
- In response to the development of OODBs, the relational database community developed Object-Relational Databases (ORDBs), which extend the relational data model with support for many of the similar object-oriented concepts.
- The concepts of Object Based Databases (OBD) are of paramount importance in today's technological development.
- Object Oriented Programming (OOP), especially in C++, Java etc., has become the dominant software-development methodology. This led to the development of an object-oriented data model. ORDBs blur the distinction between object-oriented and relational databases.
- Object Oriented Database Systems (OODBMS) are alternative to relational database and other database systems. In object oriented database, information is represented in the form of objects.
- Object oriented databases are exactly same as object oriented programming languages. If we can combine the features of relational model (transaction, concurrency, recovery) to object oriented databases, the resultant model is called as object oriented database model.
- Today's trend in programming languages is to utilize objects, thereby making OODBMS is ideal for Object Oriented programmers because they can develop the product, store them as objects, and can replicate or modify existing objects to make new objects within the OODBMS.
- Fig. 5.3 shows approaches of relational database and Object Oriented (OO) database.

COURSE

Topic	Teacher	Student	Time
Maths	Mr Salunkhe	Amar	10 am
Maths	Mr Salunkhe	Atul	10 am
...			

(a) Relational Database Approach



(b) Object-Oriented Database Approach

Fig. 5.3

Features of OODBMS:

- In OODBMS, every entity is considered as object and represented in a table. Similar objects are classified to classes and subclasses and relationship between two objects is maintained using concept of inverse reference.
- Some of the **features of OODBMS** are as follows:

1. **Complexity:** OODBMS has the ability to represent the complex internal structure (of object) with multilevel complexity.
2. **Inheritance:** Creating a new object from an existing object in such a way that new object inherits all characteristics of an existing object.

3. **Encapsulation:** It is an data hiding concept in OOPL which binds the data and functions together which can manipulate data and not visible to outside world.
4. **Persistency:** OODBMS allows to create persistent object (object remains in memory even after execution). This feature can automatically solve the problem of recovery and concurrency.

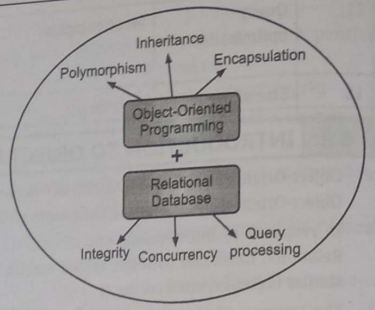


Fig. 5.4: Object-Oriented Database (Product of OOP and RDB)

Advantages of OODBMS:

1. **More Expressive Query Language:** OODBMS provides navigational access from one object to the next for data access in contrast to the associative access of SQL.
2. **Reduced Redundancy and Increased Extensibility:** OODBMS allows new data types to be built from existing types. It has the ability to factor out common properties of several classes and from them into super classes that can be shared with subclasses.
3. **Enriched Modeling Capabilities:** OODBMS allows the real-world to be modeled more closely.
4. **Removal of Impedance Mismatch:** OODBMS provides single language interface between the Data Manipulation Language (DML) and the programming language.
5. **Improved Performance:** OODBMS improves the overall performance of DBMS.
6. **Applicability to Advanced Database Applications:** The enriched modelling capabilities of OODBMSs have made them suitable for advaneved database applications such as, CAD, Office Information System (OIS), CASE, Multimedia Systems and so on.
7. **Support for Long-duration Transaction:** OODBMS uses different protocol to handle the type of long-duration transaction in contrast to enforcing serialisability on concurrent transactions by RDBMSs to maintain database consistency.
8. **Support for Schema Evolution:** In OODBMS, schema evolution is more feasible. Generalisation and inheritance allow the schema to be better structured, to be more intuitive and to capture more of the semantics of the application.

Disadvantages of OODBMS:

1. OODBMS has lack of universal data model and standards.
2. In OODBMS locking at object level may impact performance.
3. OODBMS is complex due to increased functionality.
4. OODBMS lack of support for views and also for security.

5.3 XML DATABASES

- XML stands for eXtensible Markup Language and is a text-based markup language. XML Database is used to store huge amount of information in the XML format.
- As the use of XML is increasing in every field, it is required to have a secured place to store the XML documents.
- The data stored in the database can be queried using XQuery, serialized, and exported into a desired format.

- An XML database is a data persistence software system that allows data to be specified, and sometimes stored, in XML format. An XML database is a data persistence software system that allows data to be specified, and sometimes stored, in XML format.
- There are two major types of XML databases namely, XML- enabled, Native XML (NXD).

1. **XML (Enabled Database):**

- XML enabled database is nothing but the extension provided for the conversion of XML document.
- This is a relational database, where data is stored in tables consisting of rows and columns. The tables contain set of records, which in turn consist of fields.

2. **Native XML Database:**

- Native XML database is based on the container rather than table format. It can store large amount of XML document and data. Native XML database is queried by the XPath-expressions.
- Native XML database has an advantage over the XML-enabled database. It is highly capable to store, query and maintain the XML document than XML-enabled database.

Example: Following example demonstrates XML database:

```
<?xml version = "1.0"?>
<contact-info>
   <contact1>
      <name>James Bond</name>
      <company>Target</company>
      <phone>(91)9024356173</phone>
   </contact1>
   <contact2>
      <name>Amar Salvi</name>
      <company>Nirali Prakashan</company>
      <phone>(91) 8767543902 </phone>
   </contact2>
</contact-info>
```

- Here, a table of contacts is created that holds the records of contacts (contact1 and contact2), which in turn consists of three entities, name, company and phone.

Advantages of XML Databases:

1. **Simplicity:** Information coded in XML is easy to read and understand, plus it can be processed easily by computers.
2. **Openness:** XML is a W3C standard, endorsed by software industry market leaders.
3. **Platform Independent:** XML is completely compatible with Java™ and 100% portable. Any application that can process XML can use the information, regardless of platform.
4. **Extensibility:** XML is extendable in which users can create his/her own tags.
5. **Separates Content from Presentation:** XML tags describe meaning not presentation. The look and feel of an XML document can be controlled by XSL stylesheets, allowing the look of a document (or of a complete Web site) to be changed without touching the content of the document. Multiple views or presentations of the same content are easily rendered.
6. **Facilitates the Comparison and Aggregation of Data:** The tree structure of XML documents allows documents to be compared and aggregated efficiently element by element.
7. **Can Embed Multiple Data Types:** XML documents can contain any possible data type — from multimedia data (image, sound, video) to active components (Java applets, ActiveX).

5.4 NOSQL DATABASES

- NoSQL is the acronym for Not Only SQL. NoSQL is a non-relational database management systems, different from traditional relational database management systems in some significant ways.
- NoSQL is designed for distributed data stores where very large scale of data storing needs, (for example Google or Facebook which collects terabits of data every day for their users).
- NoSQL is a new way of designing Internet-scale database solutions. It is not a product or technology but a term that defines a set of database technologies that are not based on the traditional RDBMS principles.
- NoSQL provides the process of storage and retrieval of data which is different than the relational databases. For example: NoSQL can store the data in the form of document.
- NoSQL consists of data like user information, social graphs, geographic location data and other user-generated content. Examples of NoSQL databases includes MongoDB, Cassandra etc.
- NoSQL database is an alternative to SQL database which does not require any kind of fixed table schemas unlike the SQL.
- NoSQL generally scales horizontally and avoids major join operations on the data. NoSQL database can be referred to as structured storage which consists of relational database as the subset.
- In short, NoSQL to refer to a new generation of databases that address the specific challenges of the Big Data.

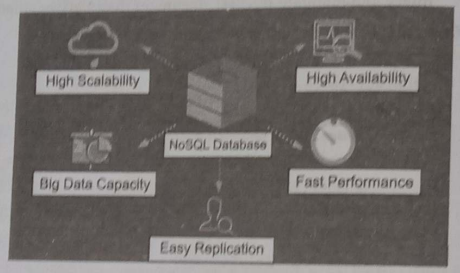


Fig. 5.5: NoSQL Databases

When to used NoSQL?

1. When huge amount of data need to be stored and retrieved.
2. The relationship between the data user store is not that important.
3. The data changing over time and is not structured.
4. Support of constraints and Joins is not required at database level.
5. The data is growing continuously and you need to scale the database regular to handle the data.

- Following is an example where it will become very difficult to store the data on RDBMS. Facebook stores terabytes of additional data every day. Let us try to imagine the structure in which this data can be structured.
- For example, the user, user's friends, who liked and Author of comments all are FB users. Now if we try to store the entire data in RDBMS, it is difficult to store all data in tabular form.

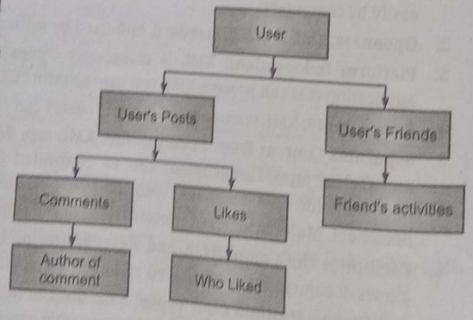


Fig. 5.6

- Hence, we need to move from tabular understanding of data to a more flow (graph) based data structure. This is what brought NoSQL structures.
- NoSQL follows CAP theorem, also called as Brewer Theorem. The CAP theorem states that no distributed database can simultaneously provide strong guarantees on the Consistency, Availability, and Partition (CAP) tolerance of a database.
- Three basic requirements stated by CAP theorem should be considered while designing a database for distributed architecture and these requirements are:
  1. **Consistency** means that the data in the database should remain consistent after any kind of transaction on database.
  2. **Availability** means that the system is always available without any down-time.
  3. **Partition tolerance** means that the system must function reliably after partitioning of server though they may not communicate with each other.

Types of NoSQL Database:

- There are four types of NoSQL databases. They are Document-Oriented Database/Document-Store Database, Graph Database/Graph-Store Database, Column-Store Database, Key-Value Store Database/Tuple-Store Database.

1. **Document-Oriented Database/Document-Store Database:**
   - Documents are the main concept in document-oriented databases in NoSQL.
   - The basic idea is in document-store databases that all the data for a single entity can be stored as a document and documents can be stored together in collections.
   - The table which contains a group of documents is called as a "collection".
2. **Graph Database/Graph-Store Database:**
   - Graph-store databases are designed for data that can be easily and simply represented as a graph. The real-world graph contains vertices and edges. They are called nodes and relations in a graph.
   - The graph databases allow us to store and perform data manipulation operations on nodes, relations and attributes of nodes and relations.
3. **Column-Store Database:**
   - Column-store databases store data in columns within a key space.
   - The key space is based on a unique name, value, and timestamp.
4. **Key-Value Store Database/Tuple-Store Database:**
   - Key-value databases store data in a completely schema-less way, meaning that no defined structure governs what is being store. A key can point to any type of data, from an object, to a string value, to a programming language function.
   - Key-value database allows the user to store data in simple <key> : <value> format, where key is used to retrieve the value from the table.

Advantages of NoSQL:

1. **High Scalability:** NoSQL database use sharding for horizontal scaling. Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding. Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement but horizontal scaling is easy to implement. Examples of horizontal scaling databases are MongoDB, Cassandra etc. NoSQL can handle huge amount of data because of scalability, as the data grows NoSQL scale itself to handle that data in efficient manner.

2. **High Availability:** NoSQL databases are generally designed to ensure high availability and avoid the complexity that comes with a typical RDBMS architecture that relies on primary and secondary nodes. NoSQL database distributed data equally among multiple resources so that the application remains available for both read and write operations even when one node fails. Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.
3. **Flexible Data Modeling:** NoSQL offers the ability to implement flexible and fluid data models. Application developers can leverage the data types and query options that are the most natural fit to the specific application use case rather than those that fit the database schema. The result is a simpler interaction between the application and the database and faster, more agile development.
4. **Low Cost:** The NoSQL databases are typically designed to work with cluster of cheap commidity servers enabling the users to store and process more data at a low cost.
5. **Better Performance:** All the NoSQL databases claim to deliver better and faster performance as compared to traditional RDBMS implementations.
6. **Variety of Data:** NoSQL databases support any type of data. It supports structured, semi-structured, and unstructured data to be stored. Its supports logs, images files, audios, videos, graphs, jpegs, JSON, XML to be stored and operated as it is without any pre-processing.
7. **Open Source:** NoSQL databases are open source. NoSQL database implementation is easy and typically uses cheap servers to manage the exploding data and transaction. So the storing and processing data cost per gigabyte in the case of NoSQL can be many times lesser than the cost of RDBMS.

Disadvantages of NoSQL:

1. **Narrow Focus:** NoSQL databases have very narrow focus as it is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.
2. **Open Source:** NoSQL is open-source database. So, there is no reliable standard for NoSQL yet. In other words two database systems are likely to be unequal.
3. **Management Challenge:** The purpose of big data tools is to make management of a large amount of data as simple as possible. But it is not so easy. Data management in NoSQL is much more complex than a relational database. NoSQL, in particular, has a reputation for being challenging to install and even more hectic to manage on a daily basis.
4. **GUI is not Available:** GUI mode tools to access the database is not flexibly available in the market.
5. **Backup:** Backup is a great weak point for some NoSQL databases like MongoDB. MongoDB has no approach for the backup of data in a consistent manner.
6. **Large Document Size:** Some database systems like MongoDB and CouchDB store data in JSON format. Which means that documents are quite large (BigData, network bandwidth, speed), and having descriptive key names actually hurts, since they increase the document size.

Difference between SQL and NoSQL database.

Sr. No.	SQL	NoSQL
1.	SQL stores data in the table.	NoSQL stores data in the form of document-store, key-value-store, graph-store etc.

Contd....

2.	SQL provides standard platform to run complex queries.	NoSQL cannot provide any standard platform to run complex queries.
3.	It supports the structured and organized data.	It supports unstructured data.
4.	Strong consistency.	Dependent on the product. Few chose to provide strong consistency whereas few provide eventual consistency.
5.	High level of enterprise support is provided.	Open source model. Support through third parties or companies building the open source products.
6.	Available through easy to use GUI interfaces.	Querying may require programming expertise and knowledge.
7.	It uses structured query language to manipulate database.	There is not declarative query language for manipulating data.
8.	SQL works on ACID (Atomicity, Consistency, Isolation and Durability) properties.	NoSQL follows CAP (Consistency, Availability and Partition tolerance) theorem.
9.	Data and its relationships are stored in separate tables.	NoSQL does not have any pre-defined schema.
10.	Examples include Oracle, SQLite, MySQL, PostgreSQL etc.	Examples include MongoDB, Cassandra, Hbase, Neo4j, BigTable etc.
11.	SQL databases are primarily called as Relational Databases (RDBMS).	NoSQL database are primarily called as non-relational or distributed database.
12.	SQL databases are vertically scalable.	NoSQL databases are horizontally scalable.
13.	SQL databases use a powerful language "Structured Query Language (SQL)" to define and manipulate the data.	In NoSQL databases, collection of documents are used to query the data. It is also called "Unstructured Query Language (UnQL)". It varies from database to database.
14.	SQL databases are best suited for complex queries.	NoSQL databases are not so good for complex queries because these are not as powerful as SQL queries.

NoSQL Using MongoDB:

- MongoDB is an open source database management system (DBMS), which is most popular NoSQL, that uses a document-oriented database model which supports various forms of data. MongoDB stores data as Binary JSON (JavaScript Object Notation) Documents (also known as BSON (Binary JSON)). The documents can have different schemas hence enabling the schema to change as the application evolves.
- Following are the reasons of MongoDB to become the most popular NoSQL database:
  1. **High Scalability:** The structure of MongoDB makes it easy to scale horizontally by sharding the data across multiple servers.
  2. **High Availability:** The replication model of MongoDB makes it easy to maintain scalability while keeping high performance and scalability.

3. **Document-Oriented:** MongoDB is document-oriented database, the data is stored in the database in a format that is very close to what we will be dealing with in both server-side and client-side scripts. This eliminates the need to transfer data from rows to objects and back.
4. **No SQL Injection:** MongoDB is not susceptible to SQL injection because objects are stored as objects, not by using SQL strings.
5. **High Performance:** It is one of the highest-performing databases available in today's world, where many people interact with websites, having a back end that can support heavy traffic is important.

5.5 MULTIMEDIA DATABASES

- Multimedia database is the collection of interrelated multimedia data that includes text, graphics (sketches, drawings), images, animations, video, audio etc. and have vast amounts of multisource multimedia data.
- The framework that manages different types of multimedia data which can be stored, delivered and utilized in different ways is known as multimedia database management system.
- There are three classes of the multimedia database which includes static media, dynamic media and dimensional media.
- A combination of multiple forms of media and content is called Multimedia. This can be text, graphics, audio, video etc.

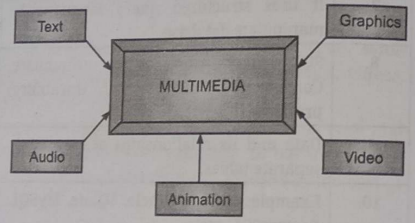


Fig. 5.7: Contents of the Multimedia Database

- The multimedia database stored the multimedia data and information related to it. This is given in detail as follows:
  1. **Media Data:** This is the multimedia data that is stored in the database such as images, videos, audios, animation etc.
  2. **Media Format Data:** The Media format data contains the formatting information related to the media data such as sampling rate, frame rate, encoding scheme etc.
  3. **Media Keyword Data:** This contains the keyword data related to the media in the database. For an image the keyword data can be date and time of the image, description of the image etc.
  4. **Media Feature Data:** The Media feature data describes the features of the media data. For an image, feature data can be colours of the image, textures in the image etc.
- A MultiMedia DataBase (MMDB) is a collection of related multimedia data. The multimedia data include one or more primary media data types such as text, images, graphic objects (including drawings, sketches and illustrations) animation sequences, audio and video etc.
- Multimedia database systems are database systems where, besides text and other discrete data, audio and video information will also be stored, manipulated and retrieved.
- Multimedia database is the collection of interrelated multimedia data that includes text, graphics (sketches, drawings), images, animations, video, audio etc. and have vast amounts of multisource multimedia data.
- There are two types of multimedia Database and they are Linked Multimedia Databases and Embedded Multimedia Database.

Advantages of Multimedia Database:

1. They support multiple formats of data (text, audio, video etc.).

2. They have flexibility of script language and Reuse of multimedia objects
3. Ability to simultaneously query different media sources and conduct classical database operations across them.

Disadvantages Multimedia Database:

1. Usually, the data size of multimedia is large such as video. Multimedia data often require a large storage.
2. Multimedia database consume a lot of processing time.
3. Production of multimedia is more expensive and costly than others because it is made up of more than one medium.

Areas where multimedia databases are applied/used:

1. **Documents and Record Management:** Industries and businesses that keep detailed records and variety of documents. Example: Insurance claim record.
2. **Knowledge Dissemination:** Multimedia database id a very effective tool for knowledge dissemination in terms of providing several resources. Example: Electronic books.
3. **Education and Training:** Computer-aided learning materials can be designed using multimedia sources which are nowadays very popular sources of learning. Example: Digital libraries.
4. **Marketing, Advertising, Retailing, Entertainment and Travel:** Example: a virtual tour of cities.
5. **Real-time Control and Monitoring:** Coupled with active database technology, multimedia presentation of information can be very effective means for monitoring and controlling complex tasks Example: Manufacturing operation control.
6. **Repository Applications** like repository of satellite images, engineering drawings, designs, space photographs and radiology scanned pictures.
7. **Presentation Applications** like our audio and video applications where data is consumed as it is delivered.
8. **Digital Library:** A digital library, digital repository or digital collection is an online database of digital objects that can include text, still images, audio, video or other digital media formats
9. **Video on Demand:** Video On-Demand (VOD) is a video media distribution system that allows users to access video entertainment without a traditional video entertainment device and without the constraints of a typical static broadcasting schedule.

5.6 BIG DATA DATABASES

- Big Data is generally considered to a very huge amount of data for storing and processing or when data itself is Big is termed as Big Data.
- Data in huge volume and different varieties can be considered as Big Data, while Database is a collection of data. We are storing data or Big Data in some type of database. So, Big Data cannot be a database. Big Data can be an entity of DB.
- Big data comes from sensors, devices, video/audio, networks, log files, transactional applications, web, and social media – much of it generated in real time and in a very large scale.

Need of Big Data:

- Big data management is the organization, administration and governance of large volumes of both structured and unstructured data.
- The goal of big data management is to ensure a high level of data quality and accessibility for business intelligence and big data analytics applications.

Definition of Big Data:

- Big Data refers to both large volumes of data with a high level of complexity and the analytical methods applied to them, which require more advanced techniques and technologies in order to derive meaningful information and insights in real time.

OR

- Big Data is high-volume, high-velocity, and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.

OR

- Big data can be defined as, "volumes of data available in varying degrees of complexity, generated at different velocities and varying degrees of ambiguity that cannot be processed using traditional technologies, processing methods, algorithms, or any commercial off-the-shelf solutions."

Processing Steps in Big Data:

- Fig. 5.8 shows a framework for big data processing that models at higher level, the working of such a system.

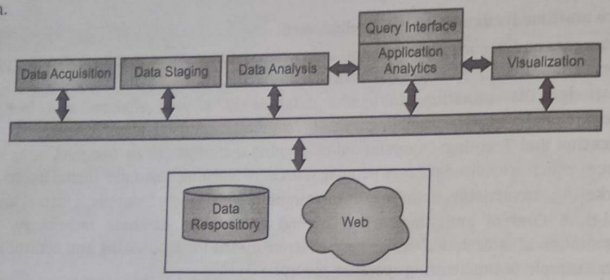


Fig. 5.8: Big Data Processing

- Big data service process has few steps starting from Data acquisition, Data staging, Data analysis and application analytics processing and visualisation.
- Source of data could be internet-based applications and databases that store organisational data. On acquiring data, preprocessing stage called data staging includes removal of unrequired and incomplete data.
- Then, it transforms data structure to a form that is required for analysis. In the process, it is most important to do data normalisation so that data redundancy is avoided.
- Normalised data then are stored for processing. Big users from different domains such as social computing, bioscience, business domains and environment to space science look forward information from gathered data.
- Analytics corresponding to an application are used for the purpose. These analytics being invoked in turn take the help of data analysis technique to scoop out information hiding in big data.
- Data analysis techniques include machine learning, soft computing, statistical methods, data mining and parallel algorithms for fast computation.
- Visualisation is an important step in big data processing. Incoming data, information while in processing and result outcome are often required to visualise for understanding because structure often holds information in its folds this is more true in genomics study.
- Big Data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data:

1. **Stock Exchange Data:** The stock exchange data holds information about the 'buy' and 'sell' decisions made on a share of different companies made by the customers.
2. **Search Engine Data:** Search engines retrieve lots of data from different databases.
3. **Black Box Data:** It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
4. **Social Media Data:** Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.

5. **Transport Data:** Transport data includes model, capacity, distance and availability of a vehicle.
6. **Power Grid Data:** The power grid data holds information consumed by a particular node with respect to a base station.

Big Data Types:

- Big data includes huge volume, high velocity and extensible variety of data. The data in it will be of three types as given below:

1. **Structured data** is concerns all data which can be stored in database SQL in table with rows and columns. They have relational key and can be easily mapped into pre-designed fields. Today, those data are the most processed in development and the simplest way to manage information.
2. **Semi-structured data** is information that does not reside in a relational database but that does have some organizational properties that make it easier and simpler to analyze. Examples of semi-structured are CSV, XML and JSON documents, NoSQL databases are considered as semi-structured.
3. **Un-structured data** refers to information that either does not have a pre-defined data model or is not organized in a pre-defined manner. Un-structured information/data is typically text-heavy, but may contain data such as dates, numbers and facts as well. Examples includes Word, PDF etc. documents.

Characteristics of Big Data:

1. **Volume:** It refers to the generation of large amount of data during data processing by using an application at every moment. For example: Twitter messages.
2. **Velocity:** It refers to the speed at which new data is generated and the speed at which data moves around the globe. For example: Stock Exchange.
3. **Variety:** It refers to the different types of data which are used in processing. The data can be structured or unstructured. For example: In face-book, user can share data in the form of text, audio, video.
4. **Veracity:** It refers to the trustworthiness of the data. Reliability data is important for organizations as well as users.

Big Data Framework (Hadoop):

- Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models.
- The Hadoop framework application works in an environment that provides distributed storageand computation across clusters of computers.
- Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Advantages of Big Data Databases:

1. **Reduced Cost:** Most big data solutions are based on the open-source technology, model and leverage commodity hardware, thus minimizing capital investments in new platforms.
2. **Economics of Scale:** A big data environment could conceivably aggregate the processing work done by multiple systems in a company into a single platform, providing not only economies of scale and process efficiencies, but reducing overall costs.
3. **Improved Customer Intelligence:** To business people, the promise of big data is really in what the integrated data types and growing data volumes about customers can mean to customer retention, customer attrition, sales uplift and revenues.
4. **Faster Processing:** With big data technologies being inherently high-housepower and parallelized faster processing means faster time to deployment.

5. **IT Transformation:** Number of IT organizations/firms especially those at older companies, recognize the promise of big data platforms to modernize outdated mainframes and antiquated code bases. One bank we know is replacing some of its core banking systems with new Hadoop platforms.

Disadvantages of Big Data Database:

1. **Data Privacy:** To ensuring that people's personal data (sensitive information) are safe from criminals and misuse.
2. **Costs:** In big data collection, aggregation, storage, analysis and reporting all cost money.
3. **Data Security:** Data security risk is obvious when considering the logistics of data collection and analysis. Data theft is a growing area of crime and attacks are getting bigger and more damaging.
4. **Bad Data:** Number of data projects a failed due to collecting irrelevant and out of date data. The real danger here is falling behind the competition. If there are not analyzing the right data and if competitors are getting it right, they take the lead.

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

1. Which database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network?
   (a) Distributed (b) Parallel
   (c) NoSQL (d) All of these
2. A database is designed to take advantage of executing operations in parallel, by running multiple instances that share a single physical database is called as?
   (a) NoSQL (b) Parallel
   (c) Distributed (d) All of these
3. Types of architectures parallel database systems includes?
   (a) Shared Memory (b) Shared Nothing
   (c) Shared Nothing (d) All of these
4. Which database is an open source and NoSQL database?
   (a) Parallel (b) Distributed
   (c) MongoDB (d) None of these
5. In which architecture, single memory is shared among many processors?
   (a) Shared Memory (b) Shared Nothing
   (c) Shared Nothing (d) All of these
6. In a _____ distributed database, all the sites use identical DBMS and operating systems.
   (a) heterogeneous (b) parallel
   (c) homogeneous (d) All of these
7. XML stands for _____.
   (a) eXparallel Markup Language (b) eXtensible Markup Language
   (c) Both (a) and (b) (d) None of these
8. Types of NoSQL includes?
   (a) Graph (b) Column
   (c) Key Value (d) All of these
9. A _____ database (MMDB) is a collection of related for multimedia data such as text, audio, graphics, video etc.
   (a) Parallel (b) Multimedia
   (c) Distributed (d) None of these

10. An _____ database is a database management system in which information is represented in the form of objects as used in Object-Oriented Programming Language (OOPL).
    (a) object (b) parallel
    (c) distributed (d) All of these
11. MongoDB is used by some of the largest companies in the world, including?
    (a) Facebook (b) Google
    (c) Cisco (d) All of these

Answers

1. (a)	2. (b)	3. (d)	4. (c)	5. (a)	6. (c)	7. (b)	8. (d)	9. (b)	10. (a)
11. (d)									

Q.II Fill in the Blanks:

1. A _____ database system seeks to improve performance through parallelization of various operations.
2. A _____ database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.
3. _____ database is a collection of multiple logically interrelated databases distributed over a computer network or Internet.
4. _____ is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.
5. A database system that is dedicated to the storage, management, and access of one or more media types, such as text, image, video, sound called as _____ database.
6. _____ is an important object-oriented feature which hides the implementation details from the end-users and displays only the needed descriptions.
7. _____ XML database is preferred over XML-enable database because it is highly capable to store, maintain and query XML documents.

Answers

1. parallel	2. NoSQL	3. distributed	4. MongoDB	5. multimedia
6. Encapsulation	7. Native			

Q.III State True or False:

1. OODBMSs also called ODBMS (Object Database Management System) combine database capabilities with Object-Oriented Programming Language (OOPL) capabilities like object, class inheritance, encapsulation etc.
2. In a distributed database the database may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers.
3. The multimedia databases are used to store multimedia data such as images, animation, audio, video along with text.
4. An XML database is a database that stores data in SQL format.
5. The process of creating new classes from existing classes is called as Inheritance which support re-usability of code.
6. Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel.
7. NoSQL Database is used to refer a non-SQL or non relational database.
8. Big data is a collection of large datasets and it is a single technique consists of various tools, techniques and frameworks like Hadoop.

9. MongoDB is one of the leading NoSQL document store platform which enables organisations to handle Big Data.

Answers

1. (T)	2. (T)	3. (T)	4. (F)	5. (T)	6. (T)	7. (T)	8. (T)	9. (T)		

Q.IV Answer the following Questions:

(A) Short Answer Questions:

1. What parallel database?
2. Define distributed database.
3. What is NoSQL?
4. Enlist purposes big data.
5. What is meant by XML data?
6. What is object based database?
7. Give two differences between parallel and distributed database?
8. What is multimedia database?
9. Comment 'why other databases are required than SQL'.
10. State two advantages and disadvantages of XML database.
11. What is meant by OODBMS?
12. Define the term distributed DBMS.
13. What is MongoDB?
14. What is CAP?
15. What is ORDBMS?
16. Enlist operations on MongoDB.

(B) Long Answer Questions:

1. Describe parallel database diagrammatically.
2. Enlist various types of distributed databases.
3. Explain the term DDBMS with its features.
4. With neat diagram distributed database.
5. Explain the following terms:
   (i) Parallel DBMS.
   (ii) Big Data databases.
6. Describe the term multimedia database diagrammatically.
7. Explain XML database with its types.
8. Describe the term big data in detail.
9. Define the following terms:
   (i) DDB
   (ii) Multimedia DBMS
   (iii) Encapsulation
   (iv) OODBMS
10. Compare multimedia and NoSQL databases.
11. With the help of diagram describe processing steps of big data.
12. Explain MongoDB with NoSQL.
14. Describe NoSQL database in detail.
15. Enlist and explain types of NoSQL databases.

❖❖❖

1. Bank Database

- Consider the following database maintained by a Bank. The Bank maintains information about its branches, customers and their loan applications.
- Following are the tables:

  branch (bid integer, brname char (30), brcity char (10))

  customer (cno integer, cname char (20), caddr char (35), city (20))

  loan_application (lno integer, lamtrequired money, lamtapproved money, l_ l_date)
- The relationship is as follows:

  branch, customer, loan_application are related with ternary relationship.

  ternary (bid integer, cno integer, lno integer).

Execution of Query:

```
postgres=# SELECT * FROM branch;
```

Output:

bid	brname	brcity
1	CAMP	PUNE
2	MGROAD	PUNE

(2 rows)

Execution of Query:

```
postgres=# SELECT * FROM customer;
```

Output:

cno	cname	caddr	city
1	KIRAN	CAMP	PUNE
2	AVINASH	MGROAD	PUNE
3	SEEMA	KOTHRUD	PUNE

(3 rows)

Execution of Query:

```
postgres=# SELECT * FROM loan_application;
```

Output:

lno	lamtrequired	lamtapproved	l_date
1	$200,000.00	$100,000.00	2016-03-01
2	$500,000.00	$300,000.00	2015-03-01
3	$300,000.00	$100,000.00	2017-02-01

(3 rows)

Execution of Query:

```
postgres=# SELECT * FROM ternary;
```

Output:

bid	cno	lno
1	1	1
2	2	2
3	3	3

(3 rows)

Solve the following Queries:

1. Find the names of the customers for the "M.G.ROAD" branch.

```
SELECT cname FROM customer a, branch b, ternary c
WHERE b.brname='MGROAD'
AND a.cno=c.cno
AND b.bid=c.bid;
```

Execution of Query:

```
postgres=# \i q1.sql
```

Output:

cname
AVINASH

(1 row)

2. List the names of the customers who have received loan less than their requirement.

```
SELECT cname FROM customer a, loan_application b, ternary c
WHERE b.lamtapproved< lamtrequired
AND a.cno=c.cno
AND b.lno=c.lno;
```

Execution of Query:

```
postgres-# \i q1.sql
```

Output:

cname
KIRAN
AVINASH

(2 rows)

3. Find the maximum loan amount approved.

```
SELECT MAX(lamtapproved) FROM loan_application ;
```

Execution of Query:

```
postgres-# \i q1.sql
```

Output:

max
$300,000.00

(1 row)

4. Find out the total loan amount sanctioned by "Camp "branch.

```
SELECT c.bid, SUM(lamtapproved) FROM branch a, loan_application b, ternary c
WHERE a.brname='CAMP'
AND a.bid=c.bid
AND b.lno=c.lno
GROUP BY c.bid;
```

Execution of Query:

```
postgres=# \i q1.sql
```

Output:

bid	sum
1	$100,000.00

(1 row)

5. Count the number of loan applications received by "M.G.ROAD" branch.

```
SELECT c.bid, COUNT(*) FROM branch a, loan_application b, ternary c
WHERE a.brname='MGROAD'
AND a.bid=c.bid
AND b.lno=c.lno
GROUP BY c.bid;
```

Execution of Query:

```
postgres=# \i q1.sql
```

Output:

bid	count
2	1

(1 row)

Views:

1. Create a view which contains the details of all customers who have applied for a loan more than ₹ 100000.

```
CREATE VIEW V1 AS
SELECT cname FROM customer a, loan_application b , ternary c
WHERE CAST(lamtrequired as numeric) > 100000
AND a.cno=c.cno
AND b.lno=c.lno
```

Execution of View:

```
postgres=# \i view1.sql
CREATE VIEW
postgres=# SELECT * FROM V1;
```

Output:

cname
KIRAN
AVINASH

(2 rows)

2. Create a view which contains details of all loan applications from the 'Shivajinagar' branch.

```
CREATE VIEW V2 AS
SELECT a.brname,b.cname,c.lamtrequired,c.lamtapproved FROM branch a, customer
                                                    b,loan_application c, ternary d
WHERE a.bid=d.bid
AND b.cno=d.cno
AND c.lno=d.lno
```

Execution of View:

```
postgres=# \i view2.sql
CREATE VIEW
```

3. Write the following Queries, on the above created views:

(a) List the details of customers who have applied for a loan of ₹ 100000.

Execution of View:

```
postgres=# SELECT * FROM V2 WHERE CAST(lamtrequired as numeric) > 100000;
```

Output:

brname	cname	lamtrequired	lamtapproved
CAMP	KIRAN	$200,000.00	$100,000.00
MGROAD	AVINASH	$500,000.00	$300,000.00

(2 rows)

(b) List the details of loan applications from 'mgroad', where loan amount is > ₹ 10000.

```
SELECT * FROM V2 WHERE brname='MGROAD'
AND CAST(lamtapproved as numeric) > 100000;
```

Execution of View:

```
postgres=# \i V2.sql
```

Output:

brname	cname	lamtrequired	lamtapproved
MGROAD	AVINASH	$500,000.00	$300,000.00

(1 row)

(c) List the details of Loan applications, with the same loan amount.

```
SELECT cname, lamtapproved FROM V2 WHERE lamtapproved IN
(SELECT lamtapproved FROM V2
GROUP BY lamtapproved
HAVING COUNT(*) > 1)
```

Execution of View:

```
postgres=# \i V2.sql
```

Output:

cname	lamtapproved
KIRAN	$300,000
SEEMA	$100,000

(2 rows)

Stored Functions:

1. Write a function that returns the total number of customers of a particular branch. (Accept branch name as input parameter).

```
CREATE OR REPLACE FUNCTION f1(name1 text) RETURNS integer AS '
DECLARE
 cnt integer;
BEGIN
SELECT INTO cnt COUNT(*) FROM branch a, customer b , ternary c
WHERE a.brname=name1
AND a.bid=c.bid
AND b.cno=c.cno;

RETURN cnt;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i f1.sql
CREATE FUNCTION
postgres=# SELECT f1('CAMP');
```

Output:

f1
1

(1 row)

Execution of Code:

```
postgres=# SELECT f1('MGROAD');
```

Output:

f1
2

(1 row)

2. Write a function to find the maximum loan amount approved.

```
CREATE OR REPLACE FUNCTION f2() RETURNS text AS '
DECLARE
 name text;
 amt money;
BEGIN
SELECT INTO name, amt a.brname, max(lamtapproved) FROM branch a, loan_application
                                                    b,ternary C
WHERE  a.bid=c.bid
AND b.lno=c.lno
GROUP BY A.brname;
name := name || amt;
RETURN name;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i f1.sql
CREATE FUNCTION
postgres=# SELECT f2();
```

Output:

f2	
MGROAD	$300,000.00

(1 row)

Errors and Exceptions:

1. Write a stored function to print the total number of customers of a particular branch.
   (Accept branch name as input parameter). In case the branch name is invalid, raise an exception for the same.

```
CREATE OR REPLACE FUNCTION f6(name1 text) RETURNS integer AS '
DECLARE
 cnt integer;
BEGIN
SELECT INTO cnt  COUNT(*) FROM branch WHERE brname=name1;
      IF cnt = 0 THEN
      RAISE EXCEPTION ''Invalid Branch Name %'', name1;
      END IF;
SELECT INTO cnt COUNT(*) FROM branch a, customer b, ternary c
WHERE a.brname=name1
AND a.bid=c.bid
AND b.cno=c.cno;
RETURN cnt;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i h2.sql
CREATE FUNCTION
postgres=# SELECT f6('pune');
```

Output:

```
ERROR:  Invalid Branch Name pune
```

2. Write a stored function to increase the loan approved amount for all loans by 20%. In case the initial loan approved amount was less than Rs 10000, then print a notice to the user, before updating the amount.

```
CREATE OR REPLACE FUNCTION f7() RETURNS integer AS '
DECLARE
amt money;
BEGIN
SELECT INTO amt lamtapproved FROM loan_application b, ternary c
WHERE b.lno=c.lno;
```



```
      IF CAST(amt as numeric) < 10000 THEN
          RAISE NOTICE ''Amount less than 10000'';
          Exit;
      END IF;
    UPDATE loan_application
    SET lamtapproved = lamtapproved + lamtapproved *0.2 ;
RETURN 0;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i u1.sql
CREATE FUNCTION
postgres=# SELECT f7();
```

Output:

f7
0

(1 row)

Execution of Code:

```
postgres=# SELECT * FROM loan_application;
```

Output:

lno	lamtrequired	lamtapproved	l_date
1	$200,000.00	$100,000.00	2016-03-01
2	$500,000.00	$300,000.00	2015-03-01
3	$300,000.00	$100,000.00	2017-02-01

(3 rows)

Triggers:

1. Write a trigger before deleting a customer record from the customer table. Raise a notice and display the message "customer record is being deleted".

```
CREATE OR REPLACE FUNCTION print_notice() RETURNS trigger AS'
DECLARE
BEGIN
    RAISE NOTICE ''deleteing Customer data ..'';
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
CREATE TRIGGER check_customers
    BEFORE DELETE  ON customer
    FOR EACH ROW
    EXECUTE PROCEDURE print_notice();
postgres=# \i t3.sql
```

```
CREATE FUNCTION
postgres=# \i t4.sql
CREATE TRIGGER
```

Output:

```
postgres=# DELETE  FROM customer WHERE cno=1;
NOTICE:  deleteing Customer data ..
DELETE 0
postgres=#
```

2. Write a trigger to ensure that the loan amount entered never < 1000 and greater than 1000000.

```
CREATE OR REPLACE FUNCTION loan_data() RETURNS trigger AS'
DECLARE
BEGIN
IF cast(NEW.lamtapproved as numeric) < 1000 or cast(NEW.lamtapproved as numeric) >
1000000 THEN
   RAISE EXCEPTION '' LOAN AMT is NOT VALID'';
   END IF;
RETURN NEW;
END;
'LANGUAGE 'plpgsql';
    CREATE TRIGGER check_loan
    BEFORE INSERT ON loan_application
    FOR EACH ROW
    EXECUTE PROCEDURE loan_data();
```

Execution of Code:

```
postgres=# \i l1.sql
CREATE FUNCTION
postgres-# \i t2.sql
CREATE TRIGGER
```

Output:

```
postgres=# INSERT INTO loan_application values(6,500,500,'2017-4-14');
ERROR: LOAN AMT is NOT VALID
postgres=#
```

2. Student-Teacher Database

- Consider a database maintained by a college and it gives information about students and the teachers along with the subject taught by the teacher and the marks obtained by the student in the subject.
- Following are the tables:

student (sno integer, s_name char(30), s_class char(10), s_addr char(50))

teacher (tno integer, t_name char (20), qualification char (15),experience integer)

- The relationship is as follows:

student-teacher: m-m with descriptive attribute as subject name and marks.

```
CREATE TABLE student
(
 sno integer PRIMARY KEY,
```



```
 sname char(30),
 sclass char(10),
 saddr char(50)
);
CREATE TABLE teacher
(
 tno integer PRIMARY KEY,
 tname char(30),
 qualification  char(15),
 experience integer
);
CREATE TABLE stud_teac
(
 sno integer references student(sno),
 tno integer  references teacher(tno),
 subject  char(30),
 marks integer
);
```

Query:

```
postgres=# \i ass2.sql
CREATE TABLE

INSERT INTO student values(1,'Amol','fybca','pune');
INSERT INTO student values(2,'Archana','fybca','pune');
INSERT INTO student values(3,'kiran','sybca','pune');
INSERT INTO student values(4,'Amar','sybca','pune');
INSERT INTO student values(5,'Suresh','tybca','pune');

INSERT INTO teacher values(1,'Kumar','PhD',10);
INSERT INTO teacher values(2,'Veena','MPhil',16);
INSERT INTO teacher values(3,'Ketaki','MCS',2);
INSERT INTO teacher values(4,'Manisha','MCA',5);
INSERT INTO teacher values(5,'Prakash','MCA',3);

INSERT INTO stud_teac values(5,1,'c',60);
INSERT INTO stud_teac values(4,1,'c++',75);
INSERT INTO stud_teac values(2,1,'c',80);
INSERT INTO stud_teac values(3,2,'DS',78);
INSERT INTO stud_teac values(4,3,'ADBMS',70);
INSERT INTO stud_teac values(5,3,'ADBMS',50);
INSERT INTO stud_teac values(1,4,'cn',66);
INSERT INTO stud_teac values(2,5,'se',73);
INSERT INTO stud_teac values(3,5,'se',85);
```

Execution of Query:

```
postgres=# SELECT * FROM student;
```

Output:

sno	sname	sclass	saddr
1	Amol	fybca	Pune
2	Archana	fybca	Pune
3	Kiran	sybca	Pune
4	Amar	sybca	Pune
5	Suresh	tybca	Pune

(5 rows)

Execution of Query:

```
postgres=# SELECT * FROM teacher;
```

Output:

tno	tname	qualification	experience
1	Kumar	PhD	10
2	Veena	MPhil	16
3	Ketaki	MCS	2
4	Manisha	MCA	5
5	Prakash	MCA	3

(5 rows)

Execution of Query:

```
postgres=# SELECT * FROM stud_teac;
```

Output:

sno	tno	subject	marks
5	1	C	60
4	1	C++	75
2	1	C	80
3	2	DS	78
4	3	ADBMS	70
5	3	ADBMS	50
1	4	CN	66
2	5	SE	73
3	5	SE	85

(9 rows)

Solve the following queries:

1. Find the minimum experienced teacher.

```
SELECT tname, experience FROM teacher
WHERE experience = (SELECT MIN(experience) FROM teacher);
```

Execution of Query:

```
postgres=# \i a3.sql
```

Output:

tname	experience
Ketki	10

(1 row)

2. Find the number of teachers having qualification "Ph. D.".

```
SELECT COUNT(*) from teacher
WHERE qualification='PhD';
```

Execution of Query:

```
postgres=# \i a3.sql
```

Output:

count
1

(1 row)

3. List the names of the students taught by "Mr. Kumar" along with the subjects taught.

```
SELECT sname, subject
FROM student a,teacher b, stud_teac c
WHERE tname='Kumar'
AND a.sno=c.sno
AND b.tno=c.tno;
```

Execution of Query:

```
postgres=# \i a3.sql
```

Output:

sname	subject
Suresh	C
Amar	C++
Archana	C

(3 rows)

4. Find the subjects taught by each teacher.

```
SELECT DISTINCT c.tno, tname, subject
FROM teacher b, stud_teac c
WHERE b.tno=c.tno
ORDER BY c.tno, tname, subject;
```

Execution of Query:

```
postgres=# \i a3.sql
```

Output:

tno	tname	subject
1	Kumar	C
1	Kumar	C++
2	Veena	DS
3	Ketaki	ADBMS
4	Manisha	CN
5	Prakash	SE

(6 rows)

5. List the names of the teachers who are teaching to a student named "Suresh".

```
SELECT tname
FROM student a, teacher b, stud_teac c
WHERE sname='Suresh'
AND a.sno=c.sno
AND b.tno=c.tno;
```

Execution of Query:

```
postgres=# \i a3.sql
```

Output:

tname
Kumar
Ketaki

(2 rows)

6. List the names of all teachers along with the total number of students they are teaching.

```
SELECT c.tno, b.tname, count(c.tno)
FROM teacher b, stud_teac c
WHERE b.tno=c.tno
GROUP BY c.tno, b.tname;
```

Execution of Query:

```
postgres=# \i a3.sql
```

Output:

tno	tname	count
5	Prakash	2
3	Ketaki	2
4	Manisha	1
1	Kumar	3
2	Veena	1

(5 rows)

7. Find the student having maximum marks in the subjects taught by "Mr. Kumar".

```
SELECT sname, marks
FROM student a, teacher b, stud_teac c
WHERE marks = (SELECT max(marks) FROM student a, teacher b, stud_teac c
   WHERE tname='Kumar'
   AND a.sno=c.sno
   AND b.tno=c.tno)
AND a.sno=c.sno;
AND b.tno=c.tno;
```

Execution of Query:

```
postgres=# \i a4.sql
```

Output:

sname	marks
Archana	80

(1 row)

Views :

1. Create a view containing details of all the teachers teaching the subject 'Mathematics'.

```
CREATE VIEW ass1 AS
SELECT distinct b.tname, b.experience
FROM teacher b, stud_teac c
WHERE subject='c'
AND b.tno=c.tno;
```

Execution of View:

```
postgres=# \i a4.sql
CREATE VIEW
postgres=# SELECT * FROM ass1;
```


Output:

tname	experience
Kumar	10

(1 row)

2. Create a view to list the details of all the students who are taught by a teacher having experience of more than three years.

```
CREATE VIEW ass3 AS
SELECT DISTINCT a.sname, a.sclass
FROM student a, teacher b, stud_teac c
WHERE a.sno=c.sno
AND b.tno=c.tno
GROUP by sname, sclass, b.experience
HAVING b.experience > 3;
```

Execution of View:

```
postgres=# \i a4.sql
CREATE VIEW
postgres=# SELECT * FROM ass3;
```

Output:

sname	sclass
Amar	sybca
Amol	fybca
Archana	fybca
Kiran	sybca
Suresh	tybca

(5 rows)

3. Write the following Queries, on the above created views:

(a) List the name of the most experienced teacher for "Mathematics".

```
SELECT MAX(experience)
FROM ass1;
```

Execution of View:

```
postgres=# \i v4.sql
```

Output:

max
10

(1 row)

(b) List the names of students of 'S.Y.B.C.A.' class, who are taught by a teacher having more than three years experience.

```
SELECT sname FROM ass3
WHERE sclass = 'sybca';
```

Execution of View:

```
postgres=# \i v4.sql
```

Output:

sname
Amar
Kiran

(2 rows)

Stored Functions:

1. Write a function to accept teacher name as input and returns the number of students taught by the teacher.

```
CREATE OR REPLACE FUNCTION tech_cnt(name text) RETURNS int AS $$
DECLARE
 cnt integer;
BEGIN
SELECT INTO cnt count(*)
FROM student a, teacher b, stud_teac c
WHERE tname= name
AND a.sno=c.sno
AND b.tno=c.tno;
RETURN cnt;
END;$$
LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i f1.sql
CREATE FUNCTION
postgres=# SELECT tech_cnt('Kumar');
```

Output:

tech_cnt
3

(1 row)

```
postgres=# SELECT tech_cnt('Veena');
```

Output:

tech_cnt
1

(1 row)

2. Write a function to accept name of subject and count the number of teachers who teach that subject.

```
CREATE OR REPLACE FUNCTION sub_cnt(name text) RETURNS int AS '
DECLARE
 no integer ;
 cnt integer;
BEGIN
SELECT INTO cnt  distinct count(tno)
FROM  stud_teac
WHERE subject= name
GROUP by tno;
RETURN cnt;
END;
'LANGUAGE 'plpgsql';
```


Execution of Code:

```
postgres=# select sub_cnt('DS');
```

Output:

sub_cnt
1

(1 row)

Execution of Code:

```
postgres=# \i f1.sql
CREATE FUNCTION
postgres=# SELECT sub_cnt('c');
```

Output:

sub_cnt
2

(1 row)

3. Write a function to accept student name and calculate the total marks obtained by that student.

```
CREATE OR REPLACE FUNCTION sub_sum(name text) RETURNS int AS '
DECLARE
 cnt integer;
BEGIN
SELECT INTO cnt sum(marks)
FROM student a, teacher b, stud_teac c
WHERE sname = name
AND a.sno=c.sno
AND b.tno=c.tno
GROUP by c.sno;
RETURN cnt;
END;
'LANGUAGE 'plpgsql';
```

Output:

```
postgres=# \i f1.sql
CREATE FUNCTION
postgres=# SELECT sub_sum('Suresh');
```

sub_sum
110

(1 row)

Cursors:

1. Write a stored function using cursors to accept student name from the user and find the names of all teachers and subjects taught to the student.

```
CREATE OR REPLACE FUNCTION tech_name(name text) RETURNS int AS '
DECLARE
 c1 cursor FOR SELECT tname, subject
FROM  student a, teacher b, stud_teac c
WHERE sname= name
AND a.sno=c.sno;
AND b.tno=c.tno;
```

```
 name1 text;
 SUB text;
 BEGIN
 OPEN c1;
 LOOP
 FETCH c1 INTO name1,sub;
 EXIT WHEN NOT FOUND;
 RAISE notice ''% %'',name1,sub;
 END LOOP;
 CLOSE c1;
 RETURN 1;
 END;
 'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# SELECT tech_name('Suresh');
```

Output:

```
NOTICE:  Kumar       c
NOTICE:  Ketaki      ADBMS
 tech_name
-----------
         1
(1 row)
```

2. Write a stored function using cursors which will calculate total number of subjects taught by each teacher.

```
CREATE OR REPLACE FUNCTION tech_cnt() RETURNS int AS '
DECLARE
 c2 cursor SELECT DISTINCT subject, tname, count(distinct(b.tno))
FROM teacher a, stud_teac b
WHERE a.tno=b.tno
GROUP by subject, tname;
name1 text;
sub1 text;
cnt integer;
BEGIN
OPEN c2;
LOOP
FETCH c2 into name1, cnt;
EXIT WHEN NOT FOUND;
RAISE NOTICE ''% % '', name1, cnt;
END LOOP;
CLOSE c2;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```


Errors and Exceptions:

1. Write a stored function to accept teacher name as input parameter and print the total number of students taught by the teacher. In case the teacher name is invalid, raise an exception for the same.

```
CREATE OR REPLACE FUNCTION tech_cnt(name text) RETURNS int AS '
DECLARE
 cnt integer;
BEGIN
SELECT INTO cnt COUNT(*)
FROM student a, teacher b, stud_teac c
WHERE tname = name
AND a.sno=c.sno
AND b.tno=c.tno;
IF cnt=0 then
RAISE EXCEPTION '' Invalid teacher '';
END IF;
RETURN cnt;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i f3.sql
CREATE FUNCTION
postgres=# SELECT tech_cnt('Kumar');
```

Output :

```
 tech_cnt
----------
        3
(1 row)
```

Execution of Code:

```
postgres=# select tech_cnt('Kk');
```

Output :

```
ERROR:   Invalid teacher
```

2. Write a stored function to increase the marks of each student to 40 if the marks are between 35 and 40. Print a notice to the user, before updating the marks.

```
CREATE OR REPLACE FUNCTION upd_mk() RETURNS int AS '
 DECLARE
 c2 CURSOR FOR SELECT sno, marks FROM stud_teac WHERE marks BETWEEN 35 AND 40;
rno integer;
mks integer;
BEGIN
OPEN c2;
LOOP
FETCH c2 into rno, mks;
EXIT WHEN NOT FOUND;
UPDATE stud_teac
SET marks=40
WHERE sno=rno;
```

```
RAISE NOTICE ''% marks are updated '',rno;
END LOOP;
CLOSE c2;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
postgres=# INSERT INTO stud_teac values(1,1,'c',35);
INSERT 0 1
postgres=# INSERT INTO stud_teac values(1,2,'DS',38);
INSERT 0 1
```

Execution of Code:

```
postgres=# \i cur1.sql
CREATE FUNCTION
postgres=# SELECT upd_mk();
```

Output :

```
NOTICE:  1 marks are updated
NOTICE:  1 marks are updated
```

upd_mk
1

(1 row)

Execution of Code:

```
postgres=# SELECT * FROM stud_teac ;
```

Output :

sno	tn0	subject	marks
5	1	C	60
4	1	C++	75
2	1	C	80
3	2	DS	78
4	3	ADBMS	70
5	3	ADBMS	50
2	5	SE	73
3	5	SE	85
1	4	CN	40
1	1	C	40
1	2	DS	40

(11 rows)

Triggers:

1. Write a trigger before deleting a student record from the student table. Raise a notice and display the message "student record is being deleted" .

```
CREATE TRIGGER  del_student
    BEFORE DELETE  ON student
    FOR EACH ROW
    EXECUTE PROCEDURE print_notice_stud();
CREATE OR REPLACE FUNCTION print_notice_stud() RETURNS trigger AS '
DECLARE
```



```
BEGIN
     RAISE NOTICE ''deleting Student data ..'';
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i c4.sql
CREATE FUNCTION
postgres=# \i trg.sql
CREATE TRIGGER
```

2. Write a trigger to ensure that the marks entered for a student, with respect to a subject is never < 0 and greater than 100.

```
CREATE OR REPLACE FUNCTION chk_stud() RETURNS trigger AS '
DECLARE
mk integer;
BEGIN
IF NEW.marks < 0 or NEW.marks > 100 then
RAISE NOTICE ''Marks should be never < 0 or Marks should be never > 100'';
END IF;
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
    CREATE TRIGGER  chk_marks
    BEFORE INSERT  ON stud_teac
    FOR EACH ROW
    EXECUTE PROCEDURE chk_stud();
```

Execution of Code:

```
postgres=# \i c5.sql
CREATE FUNCTION
postgres=# \i trg1.sql
CREATE TRIGGER
```

Output:

```
postgres=# INSERT INTO stud_teac values(1,3,'ADBMS', 105);
NOTICE:  Marks should be never < 0 or Marks should be never > 100
postgres=# INSERT INTO stud_teac values(1,3,'ADBMS', 105);
NOTICE:  Marks should be never < 0 or Marks should be never > 100
```

3. Movie Database

- Consider the following Movie database:
  movies (m_name varchar (25), release_year integer, budget money)
  actor (a_name char (30), role char (30), charges money, a_address varchar (30))
  producer(producer_id integer, name char (30), p_address varchar (30))
  Each actor has acted in one or more movies. Each producer has produced many movies and each movie can be produced by more than one producers. Each movie has one or more actors acting in it, in different roles.
- Create the above database in PostgreSQL.

```
CREATE TABLE movies
(
```

```
  mno integer PRIMARY KEY,
  mname char(30),
  ryear integer,
  budget money
 );
 CREATE TABLE actor
 (
  ano integer PRIMARY KEY,
  aname char(30),
  role  char(15),
  a_addr char(30),
 charges money
 );
 CREATE TABLE producer
 (
  pno integer PRIMARY KEY,
  pname char(30),
  p_addr char(30)
 );
 CREATE TABLE mov_act_pro
 (
  mno integer references movies(mno),
  ano integer  references actor(ano),
  pno integer  references producer(pno)
 ) ;
 INSERT INTO movies values(1,'PK',2015,1000000);
 INSERT INTO movies values(2,'piku',2014,20000000);
 INSERT INTO movies values(3,'Bajirao mastani',2016,30000000);
 INSERT INTO movies values(4,'Dangal',2016,50000000);
 INSERT INTO movies values(5,'Student of Year',2015,25000000);

 INSERT INTO actor values(1,'Amitabh','Hero', 'Mumbai',100000000);
 INSERT INTO actor values(2,'Amir','Hero','Delhi',200000000);
 INSERT INTO actor values(3,'Deepika','Heroine','Pune',10000000);
 INSERT INTO actor values(4,'Anushka','Heroine','Chennai',5000000);
 INSERT INTO actor values(5,'Aliya','Heroine','Delhi',6000000);
 INSERT INTO actor values(6,'Varun','Villan','Pune',30000000);

 INSERT INTO producer values(1,'Karan','Mumbai');
 INSERT INTO producer values(2,'Shetty','Pune');
 INSERT INTO producer values(3,'Khan','Mumbai');
 INSERT INTO producer values(4,'Salman','Mumbai');
 INSERT INTO producer values(5,'Amir','Delhi');

 INSERT INTO mov_act_pro values(2,1,1);
 INSERT INTO mov_act_pro values(2,3,2);
 INSERT INTO mov_act_pro values(1,2,5);
 INSERT INTO mov_act_pro values(1,4,5);
 INSERT INTO mov_act_pro values(3,3,4);
 INSERT INTO mov_act_pro values(3,5,4);
 INSERT INTO mov_act_pro values(4,6,3);
 INSERT INTO mov_act_pro values(4,4,5);
```

$R = \frac{6 - 2d^2}{n[n^2-1]}$

Execution of Query:

```
postgres=# SELECT * FROM movies;
```

Output:

mno	mname	ryear	budget
1	PK	2015	$1,000,000.00
2	Piku	2014	$20,000,000.00
3	Bajirao mastani	2016	$30,000,000.00
4	Dangal	2016	$50,000,000.00
5	Student of Year	2015	$25,000,000.00

(5 rows)

Execution of Query:

```
postgres=# SELECT * FROM actor;
```

Output:

ano	aname	role	a_addr	charges
1	Amitabh	Hero	Mumbai	$100,000,000.00
2	Amir	Hero	Delhi	$200,000,000.00
3	Deepika	Heroine	Pune	$10,000,000.00
4	Anushka	Heroine	Chennai	$5,000,000.00
5	Aliya	Heroine	Delhi	$6,000,000.00
6	Varun	Villan	Delhi	$30,000,000.00

(6 rows)

Execution of Query:

```
postgres=# SELECT * FROM producer;
```

Output:

pno	pname	p_addr
1	Karan	Mumbai
2	Shetty	Pune
3	Khan	Mumbai
4	Salman	Mumbai
5	Amir	Delhi

(5 rows)

Execution of Query:

```
postgres=# SELECT * FROM mov_act_pro;
```

Output:

mno	ano	pno
2	1	1
2	3	2
1	2	5
1	4	5
3	3	4
3	5	4
4	6	3
4	4	5

(8 rows)

Execute the following queries in PostgreSQL:

1. List the names of actors who have acted in at least one movie, in which Mr. Amir has acted.

```
SELECT aname, mname FROM movies a, actor b, mov_act_pro c
WHERE c.mno = (SELECT c.mno FROM movies a, actor b, mov_act_pro c
WHERE aname='Amir'
AND a.mno=c.mno
AND b.ano=c.ano)
AND a.mno=c.mno
AND b.ano=c.ano
GROUP by aname, mname
HAVING count(*)>= 1;
```

Execution of Query:

```
postgres=# \i as3.sql
```

Output:

aname	mname
Anushka	PK
Amir	PK

(2 rows)

2. List the names of the actors and their movie names.

```
SELECT aname, mname FROM movies a, actor b, mov_act_pro c
WHERE a.mno=c.mno
AND b.ano=c.ano;
```

Execution of Query:

```
postgres=# \i as3.sql
```

Output:

aname	mname
Amitabh	Piku
Amir	PK
Anushka	PK
Deepika	Bajirao mastani
Aliya	Bajirao mastani
Varun	Dangal
Anushka	Dangal

(8 rows)

3. List the names of movies whose producer is "Mr. Khan"

```
SELECT pname, mname FROM movies a, producer c, mov_act_pro d
WHERE pname='Khan'
AND a.mno=d.mno
AND c.pno=d.pno;
```

Execution of Query:

```
postgres=# \i as3.sql
```

Output:

pname	mname
Khan	Dangal

(1 row)

4. List the names of the movies with the highest budget.

```
SELECT  mname, budget
FROM movies
WHERE budget =(SELECT MAX(budget) FROM movies);
```

Execution of Query:

```
postgres=# \i as3.sql
```

Output:

mname	budget
Dangal	$50,000,000.00

(1 row)

5. List the names of movies released after 2000.

```
SELECT   mname, ryear
FROM movies
WHERE ryear > 2000;
```

Execution of Query:

```
postgres=# \i as3.sql
```

Output:

mname	ryear
PK	2015
Piku	2014
Bajirao mastani	2016
Dangal	2016
Student of the Year	2015

(5 rows)

6. List the names of actors who played the role of 'Villan'.

```
SELECT   aname, role
FROM actor
WHERE role = 'Villan';
```

Execution of Query:

```
postgres=# \i as3.sql
```

Output:

aname	role
Varun	Villan

(1 row)

7. List the names of actors who are given the maximum charges for their movie along with movie name and release year.

```
SELECT   aname, mname, ryear, charges
FROM movies  a, actor b, mov_act_pro c
WHERE  charges = (SELECT MAX(charges) FROM actor)
AND a.mno=c.mno
AND b.ano=c.ano;
```

Execution of Query:

```
postgres=# \i as3.sql
```

Output:

aname	mname	ryear	charge
Amir	PK	2015	$200,000,000.00

(1 row)

8. Display count and total budget of all movies released in year 2015.

```
SELECT   COUNT(mno), SUM(budget)
FROM movies
WHERE  ryear = 2015;
```

Execution of Query:

```
postgres=# \i as3.sql
```

Output:

count	sum
2	$26,000,000.00

(1 row)

Cursor and Triggers:

1. Write a trigger before inserting into a movie table to check budget. Budget should be minimum 50 lakh. Display appropriate message.

```
CREATE OR REPLACE FUNCTION chk_budget() RETURNS trigger AS '
DECLARE
BEGIN
IF cast(NEW.budget as numeric)  <  50000 THEN
RAISE NOTICE ''Budget should be never < 50000 '';
END IF;
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
CREATE TRIGGER  Trg_budget1
    BEFORE INSERT  ON movies
    FOR EACH ROW
    EXECUTE PROCEDURE chk_budget();
```

Execution of Code:

```
postgres=# \i as3.sql
CREATE FUNCTION
postgres=# \i trg2.sql
CREATE TRIGGER
```

Output:

```
postgres=# INSERT INTO movies values(6,'Dhoom 3', 2013, 2000);
NOTICE:  Budget should be never < 50000
INSERT 0 0
```


2. Write a stored function using cursors to display the names of actors who have acted in the maximum number of movies.

```
postgres=# INSERT INTO mov_act_pro values(4,3,2);
INSERT 0 1
```

Execution of Code:

```
postgres=# SELECT * FROM mov_act_pro;
```

Output:

mno	ano	pno
2	1	1
2	3	2
1	2	5
1	4	5
3	3	4
3	5	4
4	6	3
4	4	5
4	3	2

(9 rows)

```
CREATE OR REPLACE FUNCTION print_act() RETURNS int AS '
 DECLARE
 c2 CURSOR FOR SELECT aname, COUNT(b.mno)
FROM actor a, mov_act_pro b
WHERE a.ano=b.ano
GROUP by aname
ORDER BY COUNT(b.mno) desc
Limit 1;
name char(30);
cnt integer;
BEGIN
OPEN c2;
LOOP
FETCH c2 INTO name, cnt;
EXIT WHEN NOT FOUND;
RAISE NOTICE ''% %'', name, cnt;
END LOOP;
CLOSE c2;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i cur3.sql
CREATE FUNCTION
postgres=# SELECT print_act();
NOTICE:  Deepika  3
```

Output:

print_act
1

(1 row)

Stored Functions:

1. Write a function to list movie-wise charges of 'Amitabh Bachchan'.

```
CREATE OR REPLACE FUNCTION print_chrge(act_nm text) RETURNS int AS '
DECLARE
rec record;
BEGIN
FOR rec IN SELECT mname, charges
FROM  movies a, actor b,  mov_act_pro c
WHERE aname= act_nm
AND a.mno=c.mno
AND b.ano=c.ano
LOOP
RAISE NOTICE ''% %'',rec.mname,rec.charges;
END LOOP;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i try4.sql
CREATE FUNCTION
postgres=# SELECT print_chrge('Amitabh');
```

Output:

```
NOTICE: piku $100,000,000.00
```

print_chrge
1

(1 row)

2. Write a stored function to accept producer name as input and print the names of movies produced by him/her.. Also print the total number of actors in that movie.

```
CREATE OR REPLACE FUNCTION print_prod(act_nm text) RETURNS int AS '
DECLARE
rec record;
rec1 record;
BEGIN
FOR rec IN SELECT distinct c.mno, mname, pname
FROM  movies a,   mov_act_pro c, producer d
WHERE pname= act_nm
AND c.pno=d.pno
AND a.mno=c.mno
LOOP
FOR rec1 IN SELECT COUNT(*) as cnt FROM mov_act_pro WHERE mno=rec.mno
LOOP
```



```
RAISE NOTICE ''% %'',rec.mname,rec1.cnt;
END LOOP;
END LOOP;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i t4.sql
CREATE FUNCTION
postgres=# SELECT print_prod('Amir');
```

Output:

```
NOTICE:  PK     2
NOTICE:  Dangal 3
```

print_prod
1

(1 row)

3. Write a stored function to accept movie name as input and print the names of actors working in the movie.

```
CREATE OR REPLACE FUNCTION print_name(mov_nm text) RETURNS int AS '
DECLARE
rec record;
BEGIN
FOR rec IN SELECT aname
FROM  movies a,  actor b, mov_act_pro c
WHERE mname= mov_nm
AND b.ano=c.ano
AND a.mno=c.mno
LOOP
RAISE NOTICE ''% '',rec.aname;
END LOOP;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i t4.sql
CREATE FUNCTION
postgres=# SELECT print_name('Dangal');
```

Output:

```
NOTICE:  Varun
NOTICE:  Anushka
NOTICE:  Deepika
```

print_name
1

(1 row)

Views:

1. List the names of producers who produce the same movie as 'Mr. Karan Johar' has produced.

```
CREATE VIEW V1 AS
SELECT DISTINCT pname
FROM  movies a,   mov_act_pro c, producer d
WHERE c.mno= (SELECT DISTINCT c.mno FROM movies a, mov_act_pro c, producer d
WHERE pname= 'Karan'
AND c.pno=d.pno
AND a.mno=c.mno)
AND c.pno=d.pno;
```

Execution of View:

```
postgres=# \i v1.sql
CREATE VIEW
postgres=# SELECT * FROM v1;
```

Output:

pname
Karan
Shetty

(2 rows)

2. List the names of actors who do not live in Mumbai or Pune city.

```
CREATE VIEW V2 as
SELECT  aname
FROM  actor
WHERE a_addr NOT IN ('Mumbai','Pune');
```

Execution of View:

```
postgres=# \i v1.sql
CREATE VIEW
postgres=# SELECT * FROM v2;
```

Output:

aname
Amir
Anushka
Aliya

(3 rows)

3. List the actors in each movie.

```
CREATE VIEW V3 AS
SELECT  mname, aname
FROM  movies a,  actor b, mov_act_pro c
WHERE b.ano=c.ano
AND a.mno=c.mno
GROUP BY mname, aname
ORDER BY mname, aname;
```


Execution of Code:

```
postgres=# \i v1.sql
CREATE VIEW
postgres=# SELECT * FROM v3;
```

Output:

mname	aname
Bajirao mastani	Aliya
Bajirao mastani	Deepika
Dangal	Anushka
Dangal	Deepika
Dangal	Varun
Piku	Amitabh
Piku	Deepika
PK	Amir
PK	Anushka

(9 rows)

Exceptions:

1. Write a stored function to accept movie name as input and print the names of actors working in the movie. Also print the total number of actors working in that movie. Raise an exception for an invalid movie name.

```
CREATE OR REPLACE FUNCTION print_name1(mov_nm text) RETURNS int AS '
DECLARE
rec record;
total integer:=0;
BEGIN
SELECT INTO total COUNT(*)
FROM  movies a,  actor b, mov_act_pro c
WHERE mname= mov_nm
AND b.ano=c.ano
AND a.mno=c.mno
GROUP by aname;
IF NOT FOUND THEN
RAISE EXCEPTION ''Movie  % not found'', mov_nm;
END IF;
total :=0;
FOR rec IN SELECT aname, COUNT(*) AS cnt
FROM movies a,  actor b, mov_act_pro c
WHERE mname= mov_nm
AND b.ano=c.ano
AND a.mno=c.mno
GROUP BY aname
LOOP
RAISE NOTICE ''% %'', rec.aname, rec.cnt;
total :=total + rec.cnt;
END LOOP;
```

```
RAISE NOTICE ''Total actors=%'',total;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i t4.sql
CREATE FUNCTION
postgres=# SELECT print_name1('kll');
```

Output:

```
ERROR:  Movie  kll not found
postgres=# select print_name1('Dangal');
NOTICE:  Deepika    1
NOTICE:  Varun      1
NOTICE:  Anushka    1
NOTICE:  Total actors=3
```

print_name1
1

(1 row)

2. Write a stored function to accept producer name as input and print the names of Movies produced by him/her. Also print the total number of actors in that movie. Raise an exception for an invalid producer name.

```
CREATE OR REPLACE FUNCTION print_prod1(act_nm text) RETURNS int AS '
DECLARE
rec record;
rec1 record;
total integer :=0;
BEGIN
SELECT INTO total distinct c.mno
FROM  movies a,   mov_act_pro c, producer d
WHERE pname= act_nm
AND c.pno=d.pno
AND a.mno=c.mno;
IF NOT FOUND  THEN
RAISE EXCEPTION  ''Invalid Producer Name'';
END IF;
total:=0;
FOR rec IN SELECT distinct c.mno, mname ,pname
FROM  movies a,   mov_act_pro c, producer d
WHERE pname= act_nm
AND c.pno=d.pno
AND a.mno=c.mno
LOOP
```



```
FOR rec1 IN SELECT COUNT(*) AS cnt FROM mov_act_pro WHERE mno=rec.mno
LOOP
RAISE NOTICE ''% %'',rec.mname,rec1.cnt;
END LOOP;
END LOOP;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i t5.sql
CREATE FUNCTION
postgres=# SELECT print_prod1('gg');
```

Output:

```
ERROR:  Invalid Producer Name
postgres=# SELECT print_prod1('Amir');
NOTICE:  Dangal   3
NOTICE:  PK       2
```

print_prod1
1

(1 row)

4. Students-Marks Database

- Consider the following Student-Marks database:
  Student(rollno integer, name varchar(30), address varchar(50), class varchar(10))
  Subject(scode varchar(10), subject_name varchar(20))
  Student-Subject are related with M-M relationship with attributes marks_scored.
  Create the above database in PostgreSQL.

```
CREATE TABLE student
(
 sno integer PRIMARY KEY,
 sname char(30),
 sclass char(10),
 saddr char(50)
);
CREATE TABLE subject(
 scode varchar(10)  PRIMARY KEY,
 sub_name  varchar(30)
);
CREATE TABLE stud_sub
(
sno integer references student(sno),
scode varchar(10)  references subject(scode),
marks  integer
```

```
);
INSERT INTO stud values(1,'Amol','fybca','pune');
INSERT INTO stud values(2,'Archana','fybca','pune');
INSERT INTO stud values(3,'kiran','sybca','pune');
INSERT INTO stud values(4,'Omkar','sybca','pune');
INSERT INTO stud values(5,'Suresh','tybca','pune');
INSERT INTO stud values(6,'Reena','tybca','Mumbai');

INSERT INTO subject values('BCA-301','DS');
INSERT INTO subject values('BCA-302','ARDBMS');
INSERT INTO subject values('BCA-303','SE');
INSERT INTO subject values('BCA-304','CN');
INSERT INTO subject values('BCA-305','LAB-I');
INSERT INTO subject values('BCA-306','LAB-II');
INSERT INTO subject values('BCA-101','C');
INSERT INTO subject values('BCA-501','JAVA');

INSERT INTO stud_sub values(3,'BCA-301',60);
INSERT INTO stud_sub values(3,'BCA-302',80);
INSERT INTO stud_sub values(3,'BCA-303',50);
INSERT INTO stud_sub values(3,'BCA-304',70);
INSERT INTO stud_sub values(3,'BCA-305',86);
INSERT INTO stud_sub values(3,'BCA-306',76);
INSERT INTO stud_sub values(4,'BCA-301',60);
INSERT INTO stud_sub values(4,'BCA-302',40);
INSERT INTO stud_sub values(4,'BCA-303',50);
INSERT INTO stud_sub values(4,'BCA-304',70);
INSERT INTO stud_sub values(4,'BCA-305',46);
INSERT INTO stud_sub values(4,'BCA-306',56);
INSERT INTO stud_sub values(1,'BCA-101',60);
INSERT INTO stud_sub values(2,'BCA-101',80);
INSERT INTO stud_sub values(5,'BCA-501',70);
INSERT INTO stud_sub values(6,'BCA-501',80);
```

Execution of Query:

```
postgres=# SELECT * FROM student;
```

Output:

sno	sname	sclass	saddr
1	Amol	fybca	Pune
2	Archana	fybca	Pune
3	Kiran	sybca	Pune
4	Amar	sybca	Pune
5	Suresh	tybca	Pune
6	Reena	tybca	Mumbai

(6 rows)

Execution of Query:

```
postgres=# SELECT * FROM subject;
```


Output:

scode	sub_name
BCA-301	DS
BCA-302	ARDBMS
BCA-303	SE
BCA-304	CN
BCA-305	LAB-I
BCA-306	LAB-II
BCA-101	C
BCA-501	JAVA

(8 rows)

Execution of Query:

```
postgres=# SELECT * FROM stud_sub;
```

Output:

sno	scode	marks
3	BCA-301	60
3	BCA-302	80
3	BCA-303	50
3	BCA-304	70
3	BCA-305	86
3	BCA-306	76
4	BCA-301	60
4	BCA-302	40
4	BCA-303	50
4	BCA-304	70
4	BCA-305	46
4	BCA-306	56
4	BCA-306	56
1	BCA-101	60
2	BCA-101	80
5	BCA-501	70
6	BCA-501	80

(16 rows)

Execute the Following Queries:

1. Display the names of students scoring the maximum total marks.

```
SELECT sname, SUM(marks)
FROM student a, subject b, stud_sub c
WHERE a.sno=c.sno
AND b.scode=c.scode
GROUP BY sname
ORDER BY SUM(marks) DESC
LIMIT 1;
```

Execution of Query:

```
postgres=# \i a4.sql
```

Output:

sname	sum
Kiran	422

(1 row)

2. List the distinct names of all the subjects.

Execution of Query:

```
postgres=# SELECT * FROM subject;
```

Output:

scode	sub_name
BCA-301	DS
BCA-302	ARDBMS
BCA-303	SE
BCA-304	CN
BCA-305	LAB-I
BCA-306	LAB-II
BCA-101	C
BCA-501	JAVA

(8 rows)

3. Display class wise &subject wise student list.

```
SELECT sclass, sname FROM student
GROUP BY sclass, sname
ORDER BY sclass, sname;
```

Execution of Query:

```
postgres=# \i a4.sql
```

Output:

sclass	sname
fybca	Amol
fybca	Archana
sybca	Amar
sybca	Kiran
tybca	Reena
tybca	Suresh

(6 rows)

```
SELECT sclass,sub_name,sname
FROM student a, subject b, stud_sub c
WHERE a.sno=c.sno
AND b.scode=c.scode
GROUP BY sclass,sub_name, sname;
ORDER BY sclass,sub_name, sname;
```

Execution of Query:

```
postgres=# \i a6.sql
```


Output:

sclass	sub_name	sname
fybca	C	Amol
fybca	C	Archana
sybca	ARDBMS	Amar
sybca	ARDBMS	Kiran
sybca	CN	Amar
sybca	CN	Kiran
sybca	DS	Amar
sybca	DS	Kiran
sybca	LAB-I	Amar
sybca	LAB-I	Kiran
sybca	LAB-II	Amar
sybca	LAB-II	Kiran
sybca	SE	Amar
sybca	SE	Kiran
tybca	JAVA	Reena
tybca	JAVA	Suresh

(16 rows)

Curser and Triggers:

1. Write a stored function using cursors, to accept a address from the user and display the name, subject and the marks of the students staying at that address.

```
CREATE OR REPLACE FUNCTION print_mk(adr text) RETURNS int AS '
DECLARE
c2 CURSOR FOR SELECT sname, sub_name, marks
FROM student a, subject b, stud_sub c
WHERE saddr=adr
AND a.sno=c.sno
AND b.scode=c.scode ;
name varchar(30);
sname varchar(30);
mks  integer;
BEGIN
OPEN c2;
LOOP
FETCH c2 INTO name, sname, mks;
EXIT WHEN NOT FOUND;
RAISE NOTICE ''% %  % '',name,sname,mks;
END LOOP;
CLOSE c2;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i cur1.sql
CREATE FUNCTION
postgres=# SELECT print_mk('pune');
```

Output:

```
NOTICE:  Amol       C  60
NOTICE:  Archana    C  80
NOTICE:  kiran      DS  60
NOTICE:  kiran      ARDBMS  80
NOTICE:  kiran      SE  50
NOTICE:  kiran      CN  70
NOTICE:  kiran      LAB-I  86
NOTICE:  kiran      LAB-II  76
NOTICE:  Amar       DS  60
NOTICE:  Amar       ARDBMS  40
NOTICE:  Amar       SE  50
NOTICE:  Amar       CN  70
NOTICE:  Amar       LAB-I  46
NOTICE:  Amar       LAB-II  56
NOTICE:  Suresh     JAVA  70
 print_mk
----------
        1
(1 row)
```

Execution of Code:

```
postgres=# SELECT print_mk('Mumbai');
NOTICE:  Reena   JAVA  80
```

Output:

```
 print_mk
----------
        1
(1 row)
```

2. Write a stored function using cursors which will calculate total marks of each student.

```
CREATE OR REPLACE FUNCTION print_perc() RETURNS int AS '
DECLARE
c2 CURSOR FOR SELECT sclass, sname, sum(marks)
FROM student a, subject b, stud_sub c
WHERE  a.sno=c.sno
AND b.scode=c.scode
GROUP BY sclass, sname
ORDER BY sclass, sname;
name varchar(30);
CLASS varchar(30);
mks  integer;
BEGIN
OPEN c2;
RAISE NOTICE ''Class       Name          Total Marks'';
RAISE NOTICE ''------------------------------------------------'';
LOOP
```



```
FETCH c2 INTO class, name, mks;
EXIT WHEN NOT FOUND;
RAISE NOTICE ''% %  % '', class, name, mks;
END LOOP;
CLOSE c2;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i cur4.sql
CREATE FUNCTION
postgres=# SELECT print_perc();
```

Output:

```
NOTICE:  Class       Name          Total Marks
NOTICE:  ------------------------------------
NOTICE:  fybca       Amol          60
NOTICE:  fybca       Archana       80
NOTICE:  sybca       Amar          322
NOTICE:  sybca       kiran         422
NOTICE:  tybca       Reena         80
NOTICE:  tybca       Suresh        70
 print_perc
------------
          1
(1 row)
```

3. Write a trigger before deleting a student record from the student table. Raise a notice.and display the message "student record is being deleted".

```
CREATE OR REPLACE FUNCTION print_notice_stud() RETURNS trigger AS '
DECLARE
BEGIN
    RAISE NOTICE ''deleting Student data ..'';
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
CREATE TRIGGER  del_student2
   BEFORE DELETE  ON student
   FOR EACH ROW
   EXECUTE PROCEDURE print_notice_stud();
```

Execution of Code:

```
postgres=# \i tr.sql
CREATE FUNCTION
CREATE TRIGGER
postgres=# DELETE FROM student where sno=6;
```

Output:

```
NOTICE:  deleting Student data ..
DELETE 0
postgres=#
```

4. Write a trigger to ensure that the marks entered for a student, with respect to a subject is never < 0 and greater than 100.

```
CREATE OR REPLACE FUNCTION chk_stud1() RETURNS trigger AS '
DECLARE
BEGIN
IF NEW.marks < 0 or NEW.marks > 100 THEN
RAISE NOTICE ''Marks should be never < 0 or Marks should be never > 100'';
END IF;
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
    CREATE TRIGGER  TRG_marks
    BEFORE INSERT  ON stud_sub
    FOR EACH ROW
    EXECUTE PROCEDURE chk_stud1();
```

Execution of Code:

```
postgres=# \i trg4.sql
CREATE FUNCTION
CREATE TRIGGER
postgres=# INSERT INTO stud_sub values(3,'BCA-101',150);
```

Output:

```
NOTICE:  Marks should be never < 0 or Marks should be never > 100
INSERT 0 0
postgres=#
```

Views:

1. To list student name, class &total marks scored by each student, sorted by student name.

```
CREATE VIEW v4 AS
SELECT sname, sclass,sum(marks)
FROM student a, subject b, stud_sub c
WHERE a.sno=c.sno
AND b.scode=c.scode
GROUP BY sname, sclass
ORDER BY sname, sclass
```

Execution of View:

```
postgres=# \i a4.sql
CREATE VIEW
postgres=# SELECT * FROM v4;
```

Output:

sname	sclass	sum
Amar	sybca	322
Amol	fybca	60
Archana	fybca	80
Kiran	sybca	422
Reen	tybca	80
Suresh	tybca	70

(6 rows)

2. To list student names along with subject name and markswho scored more than 60 marks.

```
CREATE VIEW v5 AS
SELECT sname, sub_name, marks
FROM student a, subject b, stud_sub c
WHERE a.sno=c.sno
AND b.scode=c.scode
AND marks > 60
ORDER BY sname, sclass
```

Execution of View:

```
postgres=# \i a4.sql
CREATE VIEW
postgres=# SELECT * FROM v5;
```

Output:

sname	sub_name	sum
Amar	CN	70
Archana	C	80
Kiran	LAB-I	86
Kiran	LAB-II	76
Kiran	ARDBMS	80
Kiran	CN	70
Reen	JAVA	80
Suresh	JAVA	70

(8 rows)

3. Containing all the details of student named 'Amar'.

```
CREATE VIEW v8 AS
SELECT sname, sclass, saddr, sub_name, marks
FROM student a, subject b, stud_sub c
WHERE sname='Amar'
AND a.sno=c.sno
AND b.scode=c.scode;
```

Execution of View:

```
postgres=# \i a4.sql
CREATE VIEW
postgres=# SELECT * FROM v8;
```

Output:

sname	sclass	saddr	sub-name	marks
Amar	sybca	pune	DS	60
Amar	sybca	pune	ARDBMS	40
Amar	sybca	pune	SE	50
Amar	sybca	pune	CN	70
Amar	sybca	pune	LAB-I	46
Amar	sybca	pune	LAB-II	56

(6 rows)

5. Bus Transport Database

- Consider the following Bus transport Database:

  bus (bus_no int , capacity int , depot_namevar char(20))

  route (route_no int, source char(20), destination char(20), No_of_stations int)

  driver (driver_no int, driver_name char(20), license_no int, address Char(20), d_age int, salary float)

  bus_route : M-1 and Bus_Driver : M-M with descriptive attributes date of duty allotted and shift can be 1 (Morning)or 2 (Evening).

  Constraints:1. License_no is unique. 2. Bus capacity is not null.

  Create the above database in PostgreSQL.

```
CREATE TABLE route(
 route_no integer  PRIMARY KEY,
 src  varchar(30),
dest varchar(30),
no_of_station integer
);
CREATE TABLE bus
(
 bus_no integer PRIMARY KEY,
capacity integer not null,
depot_name char(30),
route_no integer references route(route_no)
);
CREATE TABLE driver
(
  dno integer PRIMARY KEY,
  dname varchar(30),
  lic_no integer unique,
  address varchar(30),
  age integer,
  salary float
);
CREATE TABLE bus_driver
(
bus_no integer references bus(bus_no),
dno  integer  references driver(dno),
date_of_duty date,
shift integer CHECK(shift in (1 ,2))
);
INSERT INTO route values(1,'Hadapsar','Katraj', 20);
INSERT INTO route values(2,'Hadapsar','Kothrud', 25);
INSERT INTO route values(3,'Hadapsar','Chinchwad', 30);
INSERT INTO route values(4,'Nigadi','Hadapsar', 30);
INSERT INTO route values(5,'Kothrud','Katraj', 22);
```



```
INSERT INTO bus values(101,35,'Hadapsar',1);
INSERT INTO bus values(102,20,'Hadapsar',2);
INSERT INTO bus values(103,32,'Hadapsar',3);
INSERT INTO bus values(104,25,'Nigadi',4);
INSERT INTO bus values(105,5,'Kothrud',5);
INSERT INTO bus values(106,25,'Kothrud',5);
INSERT INTO bus values(107,35,'Hadapsar',3);

INSERT INTO driver values(1,'Amol', 1001,'Pune',55,15000);
INSERT INTO driver values(2,'Ashok', 2001,'Karad',45,12000);
INSERT INTO driver values(3,'Bhushan', 3001,'Pune',35,10000);
INSERT INTO driver values(4,'Kiran', 4001,'Satara',25,8000);
INSERT INTO driver values(5,'Ram', 5001,'Pune',32,9000);

INSERT INTO bus_driver values(101,1,'2014-04-20',1);
INSERT INTO bus_driver values(102,2,'2014-04-20',2);
INSERT INTO bus_driver values(103,3,'2015-04-20',1);
INSERT INTO bus_driver values(104,4,'2016-04-20',2);
INSERT INTO bus_driver values(105,5,'2017-04-20',1);
INSERT INTO bus_driver values(106,3,'2015-04-20',2);
INSERT INTO bus_driver values(107,4,'2016-04-20',1);
```

Execution of Query:

```
postgres=# SELECT * FROM route;
```

Output:

route_no	src	dest	no_of_station
1	Hadapsar	Katraj	20
2	Hadapsar	Kothrud	25
3	Hadapsar	Chinchwad	30
4	Nigadi	Hadapsar	30
5	Kothrud	Katraj	22

(5 rows)

Execution of Query:

```
postgres=# SELECT * FROM bus;
```

Output:

bus_no	src	dest	route_no
101	35	Hadapsar	1
102	20	Hadapsar	2
103	32	Hadapsar	3
104	25	Nigadi	4
105	5	Kothrud	5
106	25	Kothrud	5
107	35	Hadapsar	3

(7 rows)

Execution of Query:

```
postgres=#
postgres=# SELECT * FROM driver;
```

Output:

dno	dname	lic_no	address	age	salary
1	Amol	1001	Pune	55	15000
2	Archana	2001	Pune	45	12000
3	Kiran	3001	Pune	35	10000
4	Amar	4001	Pune	25	8000
5	Suresh	5001	Pune	32	9000

(5 rows)

Execution of Query:

```
postgres=# SELECT * FROM bus_driver;
```

Output:

bus_no	dno	date_of_duty	shift
101	1	2014-04-20	1
102	2	2014-04-20	2
103	3	2015-04-20	1
104	4	2016-04-20	2
105	5	2017-04-20	1
106	3	2015-04-20	2
107	4	2016-04-20	1

(7 rows)

Execute the Following Queries:

1. Find out the name of the driver having maximum salary.

```
SELECT dname, salary
FROM driver
WHERE salary = (SELECT MAX(salary) FROM driver);
```

Execution of Query:

```
postgres=# \i as51.sql
```

Output:

dname	salary
Amol	15000

(1 row)

2. Delete the record of bus having capacity< 10

```
postgres=# DELTE FROM bus WHERE capacity < 10;
```

3. Increase the salary of all drivers by5% if driver's age> 45.

```
UPDATE driver
SET salary = salary+salary * 0.05
WHERE age > 45;
```


Execution of Query:

```
postgres=# \i as51.sql
UPDATE 1
postgres=# SELECT * FROM driver;
```

Output:

dno	dname	lic_no	address	age	salary
2	Ashok	2001	Karad	45	12000
3	Bhushan	3001	Pune	35	10000
4	Kiran	4001	Satara	25	8000
5	Ram	5001	Pune	32	9000
1	Amol	1001	Pune	55	15750

(5 rows)

4. Find out the route details on which buses of capacity 20 run.

```
SELECT a.route_no, src, dest, capacity FROM route a, bus b
WHERE b.capacity=20
AND a.route_no = b.route_no;
```

Execution of Query:

```
postgres=# \i as51.sql
```

Output:

route_no	src	dest	capacity
2	Hadapsar	Kothrud	20

(1 row)

5. Print the names and license nos. of drivers working on in both shifts.

```
SELECT dname, lic_no
FROM driver a, bus_driver b
WHERE b.shift = 1
AND a.dno = b.dno
INTERSECT
SELECT dname, lic_no
FROM driver a, bus_driver b
WHERE b.shift = 2
AND a.dno = b.dno;
```

Execution of Query:

```
postgres=# \i as51.sql
```

Output:

dname	lic_no
Kiran	4001
Bhushan	3001

(2 rows)

Cursor and Trigger:

1. Define a trigger after insert or update the record of driver if the age is between 18 and 50 give the message "valid entry" otherwise give appropriate message.

```
CREATE OR REPLACE FUNCTION chk_age() RETURNS trigger AS '
DECLARE
BEGIN
IF NEW.age > 18 and  NEW.age < 50 THEN
RAISE NOTICE ''Valid Entry'';
ELSE
RAISE EXCEPTION '' Age is not appropriate'';
END IF;
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
    CREATE TRIGGER  TRG_marks1
    AFTER INSERT  or UPDATE ON driver
    FOR EACH ROW
    EXECUTE PROCEDURE chk_age();
```

Execution of Code:

```
postgres=# \i trg5.sql
CREATE FUNCTION
CREATE TRIGGER
```

Output:

```
postgres=# INSERT INTO driver values(7,'Paul', 7001,'Pune', 17, 5000);
ERROR:  Age is not appropriate
postgres=# SELECT * FROM driver;
```

dno	dname	lic_no	address	age	salary
1	Amol	1001	Pune	55	15750
2	Ashok	2001	Karad	45	12000
3	Bhushan	3001	Pune	35	10000
4	Kiran	4001	Satara	25	8000
5	Ram	5001	Pune	32	9000

(5 rows)

2. Define a trigger after delete the record of bus having capacity< 10. Display the message accordingly.

```
CREATE OR REPLACE FUNCTION print_notice_bus() RETURNS trigger AS '
DECLARE
BEGIN
   IF (old.capacity < 10)  THEN
  RAISE NOTICE ''deleting BUS NO % data havingcapacity < 10 ...'',old.bus_no;
  END IF;
RETURN NULL;
END;
'LANGUAGE 'plpgsql';
CREATE TRIGGER  del_bus_trg3
    AFTER DELETE  ON bus
      FOR EACH ROW
    EXECUTE PROCEDURE print_notice_bus();
```


Execution of Code:

```
postgres=# \i trg5.sql
CREATE FUNCTION
CREATE TRIGGER
```

3. Write a stored function using cursors to display the details of a driver, (Accept driver name as input parameter).

```
CREATE OR REPLACE FUNCTION print_driver(name text) RETURNS int AS '
 DECLARE
 c2 CURSOR FOR select dname, lic_no, age, salary
FROM driver
WHERE dname=name;
dname varchar(30);
dlic integer;
dage  integer;
dsal float;
BEGIN
OPEN c2;
LOOP
FETCH c2 INTO dname, dlic, dage, dsal;
EXIT WHEN NOT FOUND;
RAISE NOTICE ''% %  %  %'', dname, dlic, dage, dsal;
END LOOP;
CLOSE c2;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i cur5.sql
CREATE FUNCTION
postgres=# SELECT print_driver('Amol');
```

Output:

```
NOTICE:  Amol 1001  55  15750
```

print_driver
1

(1 row)

Stored Functions:

1. Write a stored function to print the names of drivers working on both shifts on '20/04/2014'.

```
CREATE OR REPLACE FUNCTION print_dri_shift(dt date) RETURNS int AS '
DECLARE
rec record;
BEGIN
FOR rec IN SELECT dname
FROM driver a, bus_driver b
WHERE b.date_of_duty=dt
AND a.dno=b.dno
AND b.shift=1
INTERSECT
SELECT dname
FROM driver a, bus_driver b
WHERE  b.date_of_duty=dt
AND a.dno=b.dno
AND b.shift=2
LOOP
RAISE NOTICE ''%'',rec.dname;
END LOOP;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# INSERT INTO bus_driver values(106,1,'2014-04-20',2);
INSERT 0 1
postgres=# \i cur5.sql
CREATE FUNCTION
postgres=# SELECT print_dri_shift('2014-04-20');
```

Output:

```
NOTICE:  Amol
 print_dri_shift
-----------------
               1
(1 row)
```

2. Write a stored function to display the details of a driver. (Accept driver name as input parameter).

```
CREATE OR REPLACE FUNCTION print_dri_dtl(name text) RETURNS INT AS '
DECLARE
rec record;
BEGIN
FOR rec IN SELECT dname, lic_no, age, salary
FROM driver WHERE dname=name
```



```
LOOP
RAISE NOTICE ''% %  %  %'',rec.dname,rec.lic_no,rec.age,rec.salary;
END LOOP;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i cur5.sql
CREATE FUNCTION
postgres=# SELECT print_dri_dtl('Amol');
```

Output:

```
NOTICE:  Amol 1001  55  15750
 print_dri_dtl
---------------
             1
(1 row)
```

3. Write a function to accept the bus_no, date and print its allotted driver.

```
CREATE OR REPLACE FUNCTION print_dri_bus(no integer, dt date ) RETURNS int AS '
DECLARE
rec record;
BEGIN
FOR rec IN SELECT dname,bus_no
FROM driver a, bus_driver b
WHERE b.bus_no=no
AND b.date_of_duty=dt
AND a.dno=b.dno
LOOP
RAISE NOTICE ''% % '',rec.dname,rec.bus_no;
END LOOP;
RETURN 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i cur5.sql
CREATE FUNCTION
postgres=# SELECT print_dri_bus(101,'2014-04-20');
```

Output:

```
NOTICE:  Amol 101
 print_dri_bus
---------------
             1
(1 row)
```

Views:

1. Which contains details of bus no 101 along with details of all drivers who have driven that bus.

```
CREATE VIEW v51 AS
SELECT a.*
FROM driver a, bus_driver b
WHERE b.bus_no=101
AND a.dno=b.dno;
```

Execution of View:

```
postgres=# \i v5.sql
CREATE VIEW
SELECT * FROM v51;
```

Output:

dno	dname	lic_no	address	age	salary
1	Amol	1001	Pune	55	15750

(1 row)

2. To display the details of the buses that run on routes 1 or 2.

```
CREATE VIEW v512 AS
SELECT DISTINCT b.*
FROM route a, bus b
WHERE b.route_no=1 OR b.route_no=2
AND a.route_no=b.route_no;
```

Execution of Code:

```
postgres=# \i v5.sql
CREATE VIEW
SELECT * FROM v512;
```

Output:

bus_no	capacity	depot_name	route_no
102	20	Hadapsar	2
101	35	Hadapsar	1

(1 row)

3. To find out the name of the driver having maximum salary.

```
CREATE VIEW v53 AS
SELECT dname, salary
FROM driver
WHERE salary = (SELECT MAX(salary) FROM driver);
```

Execution of View:

```
postgres=# \i v5.sql
CREATE VIEW
SELECT * FROM v53;
```


Output:

dname	salary
Amol	15750

(1 row)

4. To accept the bus_no and date and print its allotted driver.

```
CREATE VIEW v54 AS
SELECT dname, bus_no
FROM driver a, bus_driver b
WHERE b.bus_no=102
AND b.date_of_duty='2014-04-20'
AND a.dno=b.dno;
```

Execution of Code:

```
postgres=# \i v5.sql
CREATE VIEW
SELECT * FROM v54;
```

Output:

dname	bus_no
Ashok	102

(1 row)

Exceptions:

1. Write a stored function to accept the bus_no and date and print its allotted drivers. Raise an exception in case of invalid bus number.

```
CREATE OR REPLACE FUNCTION print_dri_bus(no integer, dt date ) RETURNS int AS '
DECLARE
rec record;
BEGIN
SELECT * INTO rec FROM bus_driver WHERE bus_no = no AND date_of_duty=dt;
IF NOT FOUND THEN
RAISE EXCEPTION ''Invalid bus no'';
END if;
FOR rec IN SELECT dname, bus_no
FROM driver a, bus_driver b
WHERE b.bus_no=no
AND b.date_of_duty=dt
AND a.dno=b.dno
LOOP
RAISE NOTICE ''% % '', rec.dname, rec.bus_no;
END LOOP;
return 1;
END;
'LANGUAGE 'plpgsql';
```

Execution of Code:

```
postgres=# \i fun5.sql
CREATE FUNCTION
postgres=# SELECT print_dri_bus(11,'2014-04-20');
```

Output:

```
ERROR:  Invalid bus no
postgres=# SELECT print_dri_bus(101,'2014-04-20');
NOTICE:  Amol 101
 print_dri_bus
---------------
             1
(1 row)
```

Q.2
1. Explain types of cursors
2. Draw the state dig. of the transaction
3. write down use & syntex of GRANT command
4. which are the schem of recovery
5. Form concurrent enumeration
6. which are the charcteristic of big data

Q.3

Q.1
1. what do u mean by trigger
2. State the diff ways to call PLSQL fn
3. what is concuint schedule
4. Define deadlock
5. what is audit trail?
6. What do u mean by referential integrity
7. what is the use comenit comand
8. what is long
9. Define Distributed database
10. which are the type NOSQL database

❖❖❖

Syllabus ...

1. Relational Database Design Using PLSQL **(8 Hrs.)**

1.1 Introduction to PLSQL.
1.2 PL/pgSqL: Datatypes, Language Structure.
1.3 Controlling the Program Flow, Conditional Statements, Loops.
1.4 Stored Procedures.
1.5 Stored Functions.
1.6 Handling Errors and Exceptions.
1.7 Cursors.
1.8 Triggers.

2. Transaction Concepts and Concurrency Control **(10 Hrs.)**

2.1 Describe a Transaction, Properties of Transaction, State of the Transaction.
2.2 Executing Transactions Concurrently Associated Problem in Concurrent Execution.
2.3 Schedules, Types of Schedules, Concept of Serializability, Precedence Graph for Serializability.
2.4 Ensuring Serializability by Locks, Different Lock Modes, 2PL and its Variations.
2.5 Basic Timestamp Method for Concurrency, Thomas Write Rule.
2.6 Locks with Multiple Granularity, Dynamic Database Concurrency (Phantom Problem).
2.7 Timestamps versus Locking.
2.8 Deadlock and Deadlock Handling - Deadlock Avoidance (Wait-Die, Wound-Wait), Deadlock Detection and Recovery (Wait for Graph).

3. Database Integrity and Security Concepts **(6 Hrs.)**

3.1 Domain Constraints.
3.2 Referential Integrity.
3.3 Introduction to Database Security Concepts.
3.4 Methods for Database Security.
  3.4.1 Discretionary Access Control Method.
  3.4.2 Mandatory Access Control.
  3.4.3 Role Base Access Control for Multilevel Security.
3.5 Use of Views in Security Enforcement.
3.6 Overview of Encryption Technique for Security.
3.7 Statistical Database Security.

4. Crash Recovery **(4 Hrs.)**

4.1 Failure Classification.
4.2 Recovery Concepts.
4.3 Log base recovery Techniques (Deferred and Immediate Update).
4.4 Checkpoints, Relationship between Database Manager and Buffer Cache. ARIES Recovery Algorithm.
4.5 Recovery with Concurrent Transactions (Rollback, Checkpoints, Commit).
4.6 Database Backup and Recovery from Catastrophic Failure.

5. Other Databases **(2 Hrs.)**

5.1 Introduction to Parallel and Distributed Databases.
5.2 Introduction to Object Based Databases.
5.3 XML Databases.
5.4 NoSQL Database.
5.5 Multimedia Databases.
5.6 Big Data Databases.