PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE

ACADEMIC YEAR: 2019-20

LAB MANUAL

DEPARTMENT: INFORMATION TECHNOLOGY

CLASS: T.E. **SEMESTER: V**

Subject Name: Database Management System Laboratory

INDEX OF LAB EXPERIMENTS

LAB EXPT .NO	Problem Definition/Statement	Last Date Of Completion
	Group A: Introduction to Databases	
1.	Study and design a database with suitable example using following database systems: • Relational: SQL / PostgreSQL / MySQL • Key-value: Riak / Redis • Columnar: Hbase • Document: MongoDB / CouchDB • Graph: Neo4J Compare the different database systems based on points like efficiency, scalability, characteristics and performance.	22/06/2019
2.	Install and configure client and server for MySQL and MongoDB (Show all commands and necessary steps for installation and configuration).	29/06/2019
3.	Study the SQLite database and its uses. Also elaborate on building and installing of SQLite.	29/06/2019
	Group B: SQL and PL/SQL	
4.	Design & Develop DB for "Order Management System" with all the constraints. (At least 3 entities and relationships between them.) The statement should use SQL objects such as Table, View, Index, and Sequence. Draw suitable ER/EER diagram for the system. Apply DCL and DDL commands to convert ER/EER diagram to tables.	05/07/2019
5.	Manage Data into the above tables using Insert, Select, Update, Delete with operators, functions, and set operator. And Execute queries like	12/07/2019

	• Display all the Purchase orders of a specific Customer. • Get Customer and Data Item Information for a Specific Purchase Order. • Get the Total Value of Purchase Orders. • List the Purchase Orders in descending order as per total. • Display the name of customers whose first name starts with "Rav". (String matching :Like operator) • Display the name of customer whose order amount is greater than all the customers. (Relational Operator: <, >, <=, >=, = =,!=) • Display order details of customer whose city name is "Pune" and purchase date is "22/08/2016" (Boolean Operators: and, or) • Add discount of 5% to all the customers whose order is more than Rs. 10000/-. (Arithmetic Operators +, -, *, /) • Delete Purchase Order 1001.	
6.	Write following conditional select queries on above DB. A]. Aggregate functions (count, sum, avgetc) • Get the total no of customers. • Display average purchase amount of all the customers. • Display total purchase amount of all the customers. B]. Built in functions (now (), date (), day (), time () etc) • Find DAYNAME, MONTHNAME and YEAR of the purchase order made on "1995-11-2016" • Get current date & time, current time, current date • Get 6 month future & past date using interval function based on current date and name the column accordingly. • Find purchase details of the customers group by product category. • Find the purchase details of all the customers who made shopping today.(Using having clause)	20/07/2019
7.	Write following nested sub queries on above DB. A]. set membership(in, not in) • Get order details of products which are not from electronics and sports category. • Get the name and quantity of product which have either 10 or 20 or 30 quantities. B]. set comparison (<,>,<=,>=, <some, >=some, <all etc.) • Get the product details whose product price is more than "Apple 7". • Find the purchase order whose purchase amount is	27/07/2019

	greater than maximum purchase amount. Also use following keywords in nested sub queries. EXISTS /NOT EXISTS, ANY etc.	
8.	Write and execute PL/SQL block to implement all types of triggers on above DB. (Consider row level and statement level triggers)	02/08/2019
9.	Write and execute PL/SQL stored procedure and function to perform a suitable task on above DB.	10/08/2018
10.	Write and execute PL/SQL block to implement all types of cursor on above DB.	16/08/2019
11.	Write DDL statements to create VIEWS on single and multiple tables from above DB. Do the following operation to demonstrate the use of view: • Update the base table • Insert new record in the base table. • Delete record in the base table. • DML on VIEW. What are the restrictions applicable while creating or modifying views? Demonstrate using suitable queries.	23/08/2019
	Group C:MongoDB	
12.	Create a NOSQL DB on "Order management System" using MongoDB and implement following operations on document. • Insert (batch insert, insert validation) • Save • Remove • Update • Replace Document • Usage of modifiers • Upserts • Update Multiple documents • Return updated documents	31/08/2019
13.	Execute at least 10 queries on above MongoDB database that demonstrates following querying techniques: • Find • FindOne (specific values) • Conditional queries (Query conditionals, OR queries, $not, Conditional semantics) • Type-specific queries (Null, Regular expression, Querying arrays)	07/09/2019
14.	Execute at least 10 queries on above MongoDB database that	14/09/2019

	demonstrates following: • $ where queries • Cursors (Limits, skips, sorts, advanced query options) • Database commands	
15.	Implement Map reduces operation with suitable example on above MongoDB database.	21/09/2019
16.	Implement the aggregation and indexing with suitable example on above MongoDB database. Demonstrate Following • Aggregation framework • Create and drop different types of indexes and explain () to show the advantage of the indexes.	21/09/2019
	Group D: Mini Project / Database Application Development	
17.	Design and Implement any Database Application using Java/PHP/Python etc. and MySQLa s a back end. Implement Database navigation operations (add, delete, edit etc.) using ODBC/JDBC. Use stored procedure, Trigger and functions.	28/09/2018

Subject Coordinator **Head of Department**

Group A

Introduction to Databases

Assignment: 1

AIM: Study & Compare with suitable example various NoSQL database systems.

PROBLEM STATEMENT / DEFINITION:

Study and design a database with suitable example using following database systems:

- Relational: SQL / PostgreSQL / MySQL
- Key-value: Riak / Redis
- Columnar: Hbase
- Document: MongoDB / CouchDB
- Graph: Neo4J

Compare the different database systems based on points like efficiency, scalability, characteristics and performance.

OBJECTIVE:

1. To study of different type of NoSQL Databases.
2. To study of advantages of various NoSQL Databases.
3. To study of difference in NoSql and RDBMS.
4. To compare the different database systems based on points like efficiency, scalability, characteristics and performance.

THEORY:

- **What is Database?**

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds. So nowadays, we use relational database management systems (RDBMS) to store and manage huge Volume of data. This is called relational database because all the data is stored into different tables and Relations are established using primary keys or other keys known as foreign keys.

- **What is DBMS?**

A software system that enables users to define, create, maintain, and control access to the database. The DBMS is the software that interacts with the users' application programs and the database.
Typically, a DBMS provides the following facilities:

- It allows users to define the database, usually through a **Data Definition Language** (DDL). The DDL allows users to specify the data types and structures and the constraints on the data to be stored in the database.
- It allows users to insert, update, delete, and retrieve data from the database, usually through a **Data Manipulation Language (DML).** The most common query language is the Structured Query Language (SQL, pronounced 'S-Q-L', or sometimes 'See-Quel'), which is now both the formal and de facto standard language for relational DBMSs.
- It provides controlled access to the database. For example, it may provide:
  - A security system, which prevents unauthorized users accessing the database;
  - An integrity system, which maintains the consistency of stored data;
  - A concurrency control system, which allows shared access of the database;
  - A recovery control system, which restores the database to a previous consistent state following a hardware or software failure
  - A user-accessible catalog, which contains descriptions of the data in the database.

- **Advantages and Disadvantages of DBMSs**

  - Data redundancy and inconsistency
  - Difficulty in accessing data
  - Data isolation
  - Integrity Problems
  - Atomicity problem
  - Concurrent-access anomalies
  - Security Problem
  - Reduced application development time
  - Uniform data administration
  - Recovery from crashes.

- **Disadvantages**
  - Complexity
  - Size
  - Additional hardware costs
  - Cost of DBMS
  - Performance
  - Higher impact of failure

- **RDBMS Terminology:**

Before we proceed to explain MySQL database system, let's revise few definitions related to database.

- **Database:** A database is a collection of tables, with related data.
- **Table:** A table is a relation with collection of data.
- **Column:**is a field or attribute for that relation.
- **Row:** Tuple,or record) is a group of related data,
- **Primary key:** a candidate key chosen as the principal means of identifying tuples within a relation
- **Foreign key:**A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a foreign key.
- **Instance: The** collection of information stored in the database at a particular moment is called an instance of the database.
- **Schema:**The overall design of the database is called the database schema.

MySQL Database:

- MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed, and supported by MySQL AB, which is a Swedish company. MySQL isbecoming so popular because of many good reasons:
- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handleslarge subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQLworks onmanyoperatingsystemsandwithmanylanguagesincludingPHP, PERL, C,C++, JAVA,etc.
- MySQL works very quickly and works well even with large datasets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, upto50millionrowsormoreinatable. The default file size limitfor atableis4GB, but you can increase this (if your operating system can handle it) to a theoreticallimit of 8million terabytes(TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify theMySQL software to fit their own specific environments

DBMS: A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data.A DBMS makes it possible for end users to create, read, update and delete data in a database. The DBMS essentially

serves as an interface between the database and end users or application programs, ensuring that data is consistently organized and remains easily accessible.

Popular types of DBMSes:

Popular database models and their management systems include:

Relational database management system (RDMS) - adaptable to most use cases, but RDBMS Tier-1 products can be quite expensive.

NoSQL DBMS - well-suited for loosely defined data structures that may evolve over time.

In-memory database management system (IMDBMS) - provides faster response times and better performance.

Columnar database management system (CDBMS) - well-suited for da warehoatuses that have a large number of similar data items.

Cloud-based data management system - the cloud service provider is responsible for providing and maintaining the DBMS.

(I)RELATIONAL DATABASE:

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model

RDBMSs have been a common choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data, and other applications since the 1980s. Relational databases have often replaced legacy hierarchical databases and network databases because they are easier to understand and use. However, relational databases have received unsuccessful challenge attempts by object database management systems in the 1980s and 1990s (which were introduced trying to address the so-called object-relational impedance mismatch between relational databases and object-oriented application programs) and also by XML database management systems in the 1990s.

Despite such attempts, RDBMSs keep most of the market share, which has also grown over the years.

(i) SQL:

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database. This tutorial will provide you with the instruction on the basics of each of these commands as well as allow you to put them to practice using the SQL Interpreter.

(ii) PostgreSQL:

PostgreSQL, often simply Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance. As a database server, its primary functions are to store data securely and return that data in response to requests from other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications (or for data warehousing) with many concurrent users; on macOS Server, PostgreSQL is the default database; and it is also available for Microsoft Windows and Linux (supplied in most distributions).

PostgreSQL is ACID-compliant and transactional. PostgreSQL has updatable views and materialized views, triggers, foreign keys; supports functions and stored procedures, and other expandability.

PostgreSQL is developed by the PostgreSQL Global Development Group, a diverse group of many companies and individual contributors. It is free and open-source, released under the terms of the PostgreSQL License, a permissive software license.

(iii) MySQL:

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius' daughter,and "SQL", the abbreviation for Structured Query Language. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. For proprietary use, several paid editions are available, and offer additional functionality.

MySQL is a central component of the LAMP open-source web application software stack (and other "AMP" stacks). LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python". Applications that use the MySQL database include: TYPO3, MODx, Joomla, WordPress, phpBB, MyBB, and Drupal. MySQL is also used in many high-profile, large-scale websites, including Google (though not for searches), Facebook, Twitter, Flickr,and YouTube.

(II) KEY-VALUE :

A key-value store, or key-value database, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash. Dictionaries contain a collection of objects, or records, which in turn have many different fields within them, each containing data. These records are stored and retrieved using a key that uniquely identifies the record, and is used to quickly find the data within the database.

(i) Riak:

Riak is a distributed NoSQL key-value data store that offers high availability, fault tolerance, operational simplicity, and scalability.In addition to the open-source version, it comes in a supported enterprise version and a cloud storage version. Riak implements the principles from Amazon's Dynamo paper with heavy influence from the CAP Theorem. Written in Erlang, Riak has fault tolerance data replication and automatic data distribution across the cluster for performance and resilience.

- Fault-tolerant availability
- Queries
- Predictable latency
- Storage options
- Multi-datacenter replication

(ii) Redis:

Redis is an in-memory database open-source software project implementing a networked, in-memory key-value store with optional durability. Redis supports different kinds of abstract data structures, such as strings, lists, maps, sets, sorted sets, hyperloglogs, bitmaps and spatial indexes. The project is mainly developed by Salvatore Sanfilippo and is currently sponsored by Redis Labs.

(III) COLUMNAR :

A columnar database is a database management system (DBMS) that stores data in columns instead of rows. The goal of a columnar database is to efficiently write and read data to and from hard disk storage in order to speed up the time it takes to return a query.

(i) Hbase:

HBase is an open source, non-relational, distributed database modeled after Google's Bigtable and is written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System), providing Bigtable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).

(IV) DOCUMENT :

A document-oriented database, or document store, is a computer program designed for storing, retrieving and managing document-oriented information, also known as semi-structured data.

(i) MongoDB:

MongoDB (from humongous) is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. MongoDB is developed by MongoDB Inc. and is free and open-source, published under a combination of the GNU Affero General Public License and the Apache License.

(ii) CouchDB:

Apache CouchDB is open source database software that focuses on ease of use and having an architecture that "completely embraces the Web".[2] It has a document-oriented NoSQL database architecture and is implemented in the concurrency-oriented language Erlang; it uses JSON to store data, JavaScript as its query language using MapReduce, and HTTP for an API.

(V) GRAPH :

In computing, a graph database is a database that uses graph structures for semantic queries with nodes, edges and properties to represent and store data. A key concept of the system is the graph (or edge or relationship), which directly relates data items in the store.

(i) Neo4J:

Neo4j is a graph database management system developed by Neo Technology, Inc. Described by its developers as an ACID-compliant transactional database with native graph storage and processing, Neo4j is the most popular graph database according to db-engines.com.

Neo4j is available in a GPL3-licensed open-source "community edition", with online backup and high availability extensions licensed under the terms of the Affero General Public License. Neo also licenses Neo4j with these extensions under closed-source commercial terms.

Neo4j is implemented in Java and accessible from software written in other languages using the Cypher Query Language through a transactional HTTP endpoint, or through the binary 'bolt'

REFERENCE BOOK:

1. Complete Reference of MySql
2. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw HillPublishers, ISBN 0-07-120413-X

CONCLUSION:

1. Study of various open Source RDBMS & NoSQL database systems
2. Compare basic RDBMS & NoSQL database systems
3. Compare the different database systems based on points like efficiency, scalability, characteristics and performance.

Assignment: 2

AIM: Install and configure client and server for MySQL and MongoDB (Show all commands and necessary steps for installation and configuration).

PROBLEM STATEMENT / DEFINITION:

Installation and configuration of client and server for :

- MySQL (RDBMS)
- MondoDB (NoSQL)

OBJECTIVE:

1. To study installation & configuration of MySQL database.
2. To study installation & configuration of MongoDB.
3. To analyze difference between RDBMS & NoSQL installation & configurations.

THEORY:

Installation of MySQL:

First, remove the current version of MySQL you're already using:

```
$ sudo apt-get purge mysql-client-core-5.5
```

Now, to install MySQL, run the following command from a terminal prompt:

```
$ sudo apt-get install mysql-server
$ sudo apt-get install mysql-client
```

During the installation process you will be prompted to enter a password for the MySQL root user.

Once the installation is complete, the MySQL server should be started automatically. You can run the following command from a terminal prompt to check whether the MySQL server is running:

```
$ sudo netstat -tap | grep mysql
```

When you run this command, you should see the following line or something similar:

```
tcp        0      0 localhost.localdomain:mysql         *:* LISTEN -
```

If the server is not running correctly, you can type the following command to start it:

```
$ sudo /etc/init.d/mysql restart
```

You can edit the /etc/mysql/my.cnf file to configure the basic settings: log file, port number, etc.

Installation of MongoDB:

Step 1 — Importing the Public Key

In this step, we will import the MongoDB GPG public key.

MongoDB is already included in Ubuntu package repositories, but the official MongoDB repository provides most up-to-date version and is the recommended way of installing the software. Ubuntu ensures the authenticity of software packages by verifying that they are signed with GPG keys, so we first have to import they key for the official MongoDB repository.

To do so, execute:

```
$ sudo -E apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
```

After successfully importing the key you will see:

Output

```
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

Step 2 — Creating a List File

Next, we have to add the MongoDB repository details so APT will know where to download the packages from.

Issue the following command to create a list file for MongoDB.

```
$ echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb_release -sc)"/mongodb-org/3.0 multiverse" | sudo
```

```
tee /etc/apt/sources.list.d/mongodb-org-3.0.list
```

After adding the repository details, we need to update the packages list.

```
$ sudo apt-get update
```

Step 3 — Installing and Verifying MongoDB

Now we can install the MongoDB package itself.

```
$ sudo apt-get install -y mongodb-org
```

This command will install several packages containing latest stable version of MongoDB along with helpful management tools for the MongoDB server.

After package installation MongoDB will be automatically started. You can check this by running the following command.

```
$ sudo service mongod status
```

```
$ sudo service mongod start
```

If MongoDB is running, you'll see an output like this (with a different process ID).

Output

```
mongod start/running, process 1611
```

REFERENCE URL:

1. https://dev.mysql.com/doc/mysql-apt-repo-quick-guide/en/
2. https://docs.mongodb.com/v3.0/tutorial/install-mongodb-on-ubuntu/

CONCLUSION:

1. Study of installation steps on client server MySQL & MongoDB.
2. Study of configuration of MySQL & MongoDB.

Assignment: 3

AIM: Study the SQLite database and its uses and installation.

PROBLEM STATEMENT / DEFINITION:

1. Study the SQLite database and its uses.
2. Elaborate on building and installing of SQLite.

OBJECTIVE:

1. To study SQLite database and its uses.
2. To study installation & configuration of SQLite database.

THEORY:

SQLite:

SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. SQLite is the most used database engine in the world.

SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

SQLite is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax that does not guarantee the domain integrity.[5]

SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems (such as mobile phones), among others.[6] SQLite has bindings to many programming languages.

Installing SQLite:

1. type in the following command –

$ sudo apt-get install sqlite3 libsqlite3-dev

2. After installation check installation, sqlite terminal will give you a prompt and version info –

naved@neo:~$ sqlite3

SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>

3. To quit –

sqlite> .quit

4. Go to desired folder and create database –

naved@neo:~$ sqlite3 database_name.db

It'll create database_name.db in the folder you've given the command.

5. To check whether the database has been created give the following command in sqlite3 terminal –

sqlite> .databases

Uses of SQLite:

SQLite is not directly comparable to client/server SQL database engines such as MySQL, Oracle, PostgreSQL, or SQL Server since SQLite is trying to solve a different problem.

Client/server SQL database engines strive to implement a shared repository of enterprise data. They emphasis scalability, concurrency, centralization, and control. SQLite strives to provide local data storage for individual applications and devices. SQLite emphasizes economy, efficiency, reliability, independence, and simplicity.

SQLite does not compete with client/server databases.

- Embedded devices and the internet of things
- Application file format
- Websites
- Data analysis
- Cache for enterprise data
- Server-side database
- File archives
- Replacement for *ad hoc* disk files
- Internal or temporary databases
- Stand-in for an enterprise database during demos or testing
- Education and Training
- Experimental SQL language extensions

REFERENCE URL:

1. https://sysads.co.uk/2014/08/05/install-sqlite-database-browser-3-2-0-on-ubuntu-14-04/
2. http://www.sqlitetutorial.net/

CONCLUSION:

1. Study of installation steps of SQLite database.
2. Study of configuration of SQLite database.
3. To understand various uses of SQLite database.

Group B

SQL and PL/SQL

Assignment: 4

AIM:

Design & Develop DB for "Order Management System" with all the constraints

PROBLEM STATEMENT / DEFINITION:

Design & Develop DB for "Order Management System" with all the constraints. (There must be At least 3 entities and relationships between them.)

The statement should use SQL objects such as Table, View, Index, and Sequence.

Draw suitable ER/EER diagram for the system.

Apply DCL and DDL commands to convert ER/EER diagram to tables.

OBJECTIVE:

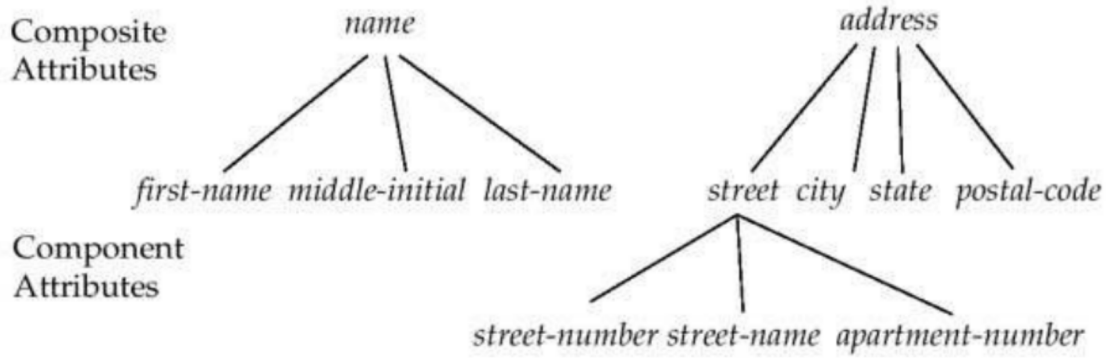
1. To understand the concept of ER diagram.
2. To understand the details of basic ER model
3. To understand the technique for converting ER diagram into tables
4. Analyze the reflected relationship and constraints
5. To understand use of DDL , DCL

THEORY:

Basic concepts of ER Diagram:
A database can be modeled as a collection of entities and relationship among Entities. Entity: entity is an object that exists and is distinguishable from other objects. Example: specific person, company, event, plant

Entity set: An entity set is a set of entities of the same type that share the same properties.
Example: set of all persons, companies, trees, holidays

Attributes: Entities have attributes Example: people have names and addresses
Attribute types:
1 Simple: e.g. roll no
2 Composite attributes: e.g. name, address
3 Single-valued: roll no
4 Multi-valued attributes: e.g. Phone-numbers
5 Derived attributes: Can be computed from other attributes
E.g. age, given date of birth



Relationship: A relationship is an association among several entities
Example: Hayes depositor A-102customer entity relationship set account entity

Relationship set: A relationship set is a mathematical relation among n ≥ 2 entities, each taken from entity sets
{(e1, e2, en) | e1 ε E1, e2 ε E2, en ε En} where (e1, e2, en) is a relationship
Example: (Hayes, A-102) ε depositor

Mapping Cardinalities:
Express the number of entities to which another entity can be associated via arelationship set. Most useful in describing binary relationship sets. For a binaryrelationship set the mapping cardinality must be one of the following types:
1. One to one 2. One to many 3. Many to one 4. Many to many

Symbolic notations:
Components to draw entity relationship diagram.
Rectangles: represent entity sets.
Diamonds: represent relationship sets.
Lines: link attributes to entity sets and entity sets to relationship sets.
Ellipses: represent attributes

Double ellipses: represent multivalued attributes.
Dashed ellipses: denote derived attributes.
Underline: indicates primary key attributes (will study later)

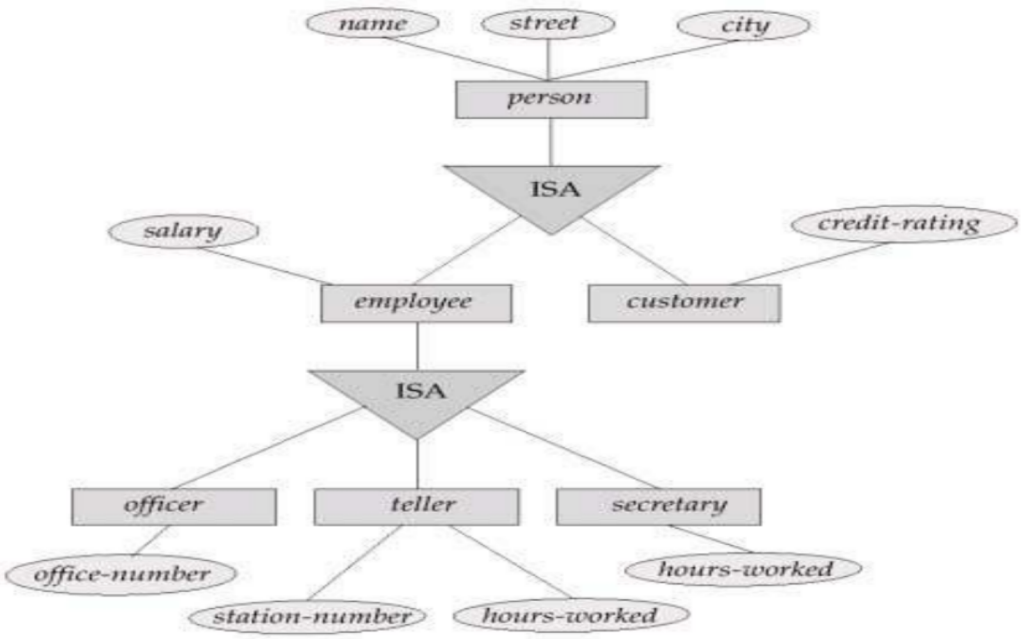
Extended ER Features:

Specialization

Top-down design process; we designate sub groupings within an entity set that are distinctive from other entities in the set. These sub groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher- level entity set. Depicted by a triangle component labeled ISA (E.g. customer "is a" person).

Attribute inheritance – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked

1) To convert an ER Diagram into Database tables.

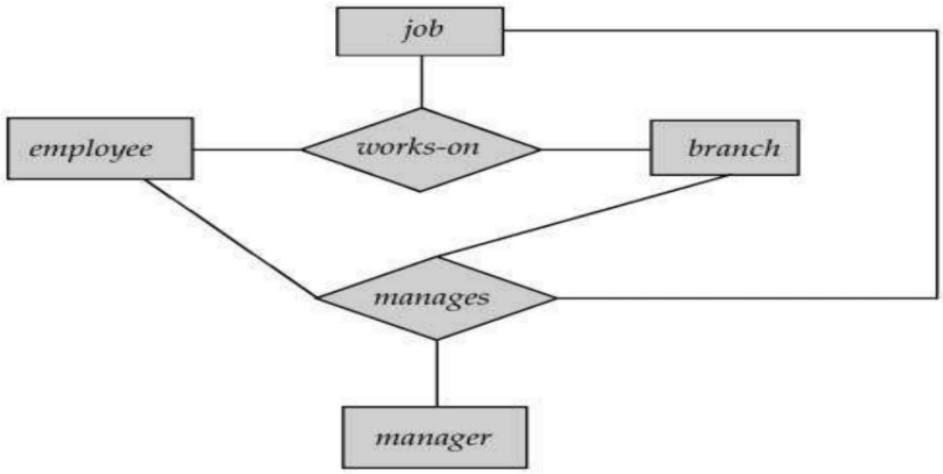


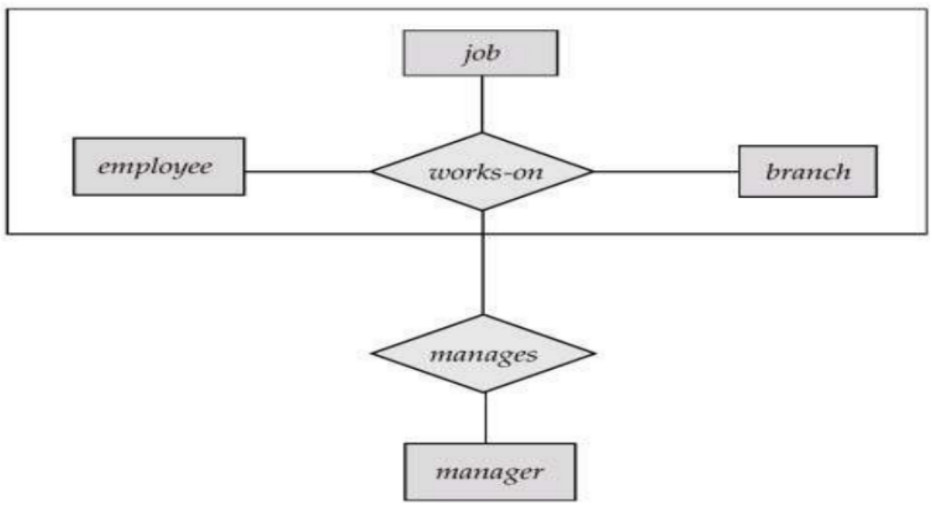
Generalization

A bottom-up design process – combine a number of entities sets that share the same features into a higher-level entity set. Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way. The terms specialization and generalization are used interchangeably.

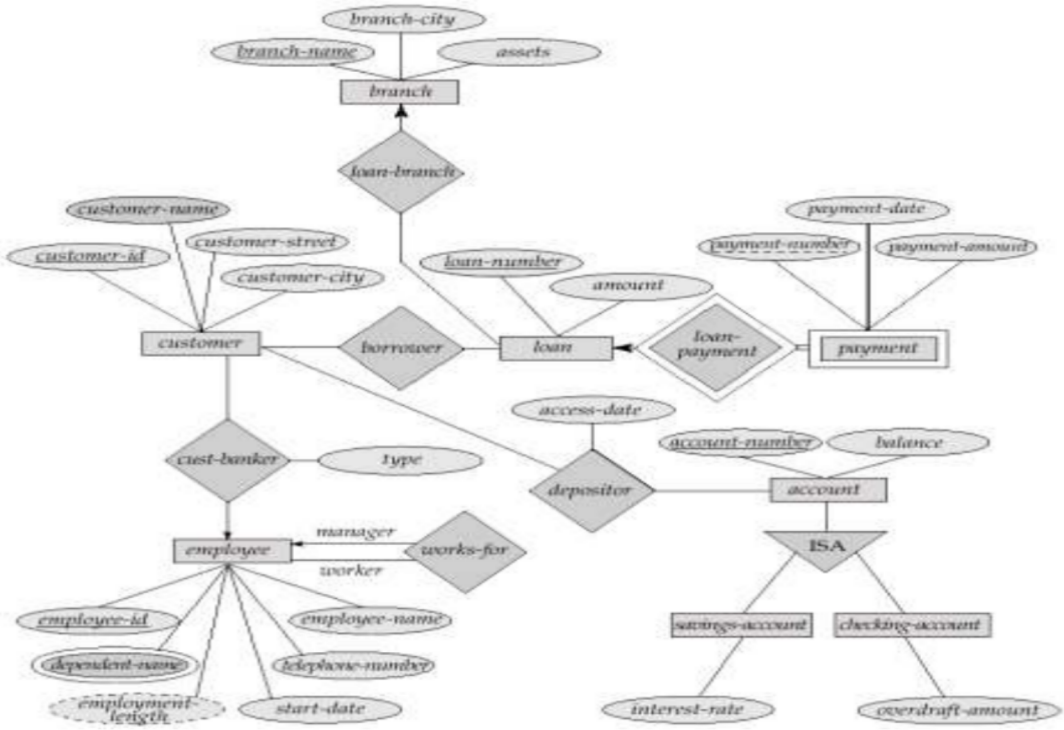
Aggregation

Consider the ternary relationship works-on, which we saw earlier. Suppose we want to record managers for tasks performed by an employee at a branchRelationship sets works-on and manages represent overlapping information. Every manages relationship corresponds to a works-on relationship However, some works-on relationships may not correspond to any manages relationships. So we can't discard the works-on relationship. Eliminate this redundancy via aggregation. Treat relationship as an abstract entity. Allows relationships between relationships. Abstraction of relationship into new entity without introducing redundancy, the following diagram represents:
An employee works on a particular job at a particular branch an employee, branch, and job combination may have an associated manager.





4)Example: E-R Diagram for Bank organization



5)Reduction of ER Schema to tables

Primary keys allow entity sets and relationship sets to be expressed uniformly as tables which represent the contents of the database. A database which conforms to an E-R diagram can be represented by a collection of tables. For each entity set and relationship set there is a unique table which is assigned the name of the corresponding entity set or relationship set. Each table has a number of columns (generally corresponding to attributes), which have unique names.Converting an E-R diagram to a table format is the basis for deriving a relational database design from an E-R diagram.A strong entity set reduces to a table with the same attributes.

Ex. Customer (customer-id, customer-name, customer-street, customer-city)Schema can be reduce as follows.

customer-id	*customer-name*	*customer-street*	*customer-city*
019-28-3746	Smith	North	Rye
182-73-6091	Turner	Putnam	Stamford
192-83-7465	Johnson	Alma	Palo Alto
244-66-8800	Curry	North	Rye
321-12-3123	Jones	Main	Harrison
335-57-7991	Adams	Spring	Pittsfield
336-66-9999	Lindsay	Park	Pittsfield
677-89-9011	Hayes	Main	Harrison
963-96-3963	Williams	Nassau	Princeton

Introduction to SQL:

The Structured Query Language (SQL) comprises one of the fundamental building blocks of modern database architecture. SQL defines the methods used to create and manipulate relational databases on all major platforms. SQL comes in many flavors. Oracle databases utilize their proprietary PL/SQL. Microsoft SQL Server makes use of Transact-SQL. However, all of these variations are based upon the industry standard ANSI SQL.
SQL commands can be divided into two main sublanguages.

1. Data Definition Language
2. Data Manipulation Language

1.1 DATA DEFINITION LANGUAGE (DDL)

It contains the commands used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.
DDL Commands:

a) Create table command:

Syntax :

CREATE TABLE table name (column_name1 data type (size), column_name2 data_type(size), ...)

Example 1

This example demonstrates how you can create a table named “Person", with four columns. The column names will be "LastName", "FirstName", "Address", and "Age":
CREATE TABLE Person (LastName varchar, FirstName varchar, Address varchar, Age int)
This example demonstrates how you can specify a maximum length for some columns:

Example 2

CREATE TABLE Person (LastName varchar(30), FirstName varchar, Address varchar, Age int(3))
Creating table from another (existing table) table: Syntax
CREATE TABLE tablename [(columnname,column name)]] AS SELECT columnname,columnname FROM tablename;

b. Alter table command:

Once table is created within a database, we may wish to modify the definition of that table.The ALTER command allows making changes to the structure of a table without deleting and recreating it.Syntax
ALTER TABLE table_name ADD (newcolumn_name1 data_type(size), newcolumn_name2 data_type(size), ...)

Example

ALTER TABLE personal_info ADD salary money null

This example adds a new attribute to the personal_info table -- an employee's salary. The "money" argument specifies that an employee's salary will be stored using a dollars and cents format. Finally, the "null" keyword tells the database that it's OK for this field to contain no value for any given employee.

c. Drop table command:

DROP command allows us to remove entire database objects from our DBMS. For example, if we want to permanently remove the personal_info table that we created, we'd use the following command:

Syntax

DROP TABLE table_name;

Example

DROP TABLE personal_info;

DATA INTEGRITY:

Enforcing data integrity ensures the quality of the data in the database. For example, if an employee is entered with an employee_id value of "123", the database should not allow another employee to have an ID with the same value. Two important steps in planning tables are to identify valid values for a column and to decide how to enforce the integrity of the data in the column. Data integrity falls into four categories:

- Entity integrity
- Domain integrity
- Referential integrity
- User-defined integrity

There are several ways of enforcing each type of integrity.

Integrity type	Recommended options
Entity	PRIMARY KEY constraintUNIQUE constraint
Domain	FOREIGN KEY constraint CHECK constraint NOT NULL
Referential	FOREIGN KEY constraint CHECK constraint
User-defined	All column- and table-level constraints in CREATE TABLE StoredProcedures Triggers

ENTITY INTEGRITY:

Entity integrity defines a row as a unique entity for a particular table. Entity integrity enforces the integrity of the identifier column(s) or the primary key of a table (through indexes, UNIQUE constraints, PRIMARY KEY constraints, or IDENTITY properties).

DOMAIN INTEGRITY:

Domain integrity is the validity of entries for a given column. You can enforce domain integrity by restricting the type (through data types), the format (through CHECK constraints and rules), or the range of possible values (through FOREIGN KEY constraints, CHECK constraints, DEFAULT definitions, NOT NULL definitions, and rules).

REFERENTIAL INTEGRITY:

Referential integrity preserves the defined relationships between tables when records are entered or deleted. In Microsoft® SQL Server™, referential integrity is based on relationships between foreign keys and primary keys or between foreign keys and unique keys. Referential integrity ensures that key values are consistent across tables. Such consistency requires that there be no references to nonexistent values and that if a key value changes, all references to it change consistently throughout the database.

PRIMARY KEY CONSTRAINT:

Definition: - The primary key of a relational table uniquely identifies each record in the table. A primary key constraint ensures no duplicate values are entered in particular columns and that NULL values are not entered in those columns.

a. **NOT NULL CONSTRAINT:**
This constraint ensures that NULL values are not entered in those columns.

b**. UNIQUE CONSTRAINT**:
This constraint ensures that no duplicate values are entered in those columns.

c. **CHECK CONSTRAINT:**
The CHECK constraint enforces column value restrictions. Such constraints can restrict a column, for example, to a set of values, only positive numbers, or reasonable dates.not working in mysql.

d. **FOREIGN KEY CONSTRAINT**:

Foreign keys constrain data based on columns in other tables. They are called foreign keys because the constraints are foreign--that is, outside the table. For example, suppose a table contains customer addresses, and part of each address is a United States two-character state code. If a table held all valid state codes, a foreign key constraint could be created to prevent a user from entering invalid state codes.

To create a table with different types of constraints:
CREATE TABLE table_name (column_name1 data_type [constraint], column_name2 data_type [constraint], ...)
Example
All Basic commands of MySql .Like :
mysql> create database ManageCust;
Query OK, 1 row affected (0.00 sec) // to user ur own database,
other than default.
mysql> use ManageCust;
Database changed
mysql> QUIT To exit the MySQL Shell, just type QUIT or EXIT:
mysql> exit
mysql> SHOW TABLES;
mysql> DESCRIBE <Table Name>;

INPUT:

Initial database is blank now consider the real-time scenario to create database management system.

1. Draw an E-R Diagram by considering notation of E-R.
2. Convert E-R to table by applying rules of conversion.
3. Design a database with DDL, DML, and DQL.

For implementation of DDL, DML, DQL statement using MySql, we have considered a real timeexample of **"Managing customer orders"**system. Following is the Scenario:

1. A customer has a unique customer number and contact information,
2. A customer can place many orders, but a given purchase order is placed by onecustomer
3. A purchase order has a many-to-many relationship with a stock item.

A)Create Table **Customer** (CustnoInt Not Null ,Custname Varchar2(200) Not Null,

Street Varchar2(200) Not Null, City Varchar2(200) Not Null,State Char(4) Not Null Default 'Pune', Zip Varchar2(20),***Primary Key (Custno)***);

B) Create Table **Purchaseorder**(PonoInt , CustnoInt , Orderdate Date, Shipdate Date, Tostreet Varchar2(200), Tocity Varchar2(200), Tostate Char(2), Tozip Varchar2(20), ***Primary Key(Pono)*** , ***Foreign Key Fk_Cust(Custno) References Customer (Custno)***) ;

C) Create Table **Contains** (PonoInt, StocknoInt**,** Quantity Int, Discount Int, ***Foreign Key Fk_Pur(Pono)References Purchaseorder (Pono)***, ***Foreign Key Fk_Stock(***Stockno***) References Stock (***Stockno***),Primary Key (Pono, Stockno)***) ;

D) Create Table **Cust_Phones**(Custno Number, Phones Varchar2 (20), ***Foreign Key Fk_Cust(Custno) References Customer (Custno)*** , ***Primary Key (Custno, Phones)***) ;

E) Create Table **Stock** (StocknoInt, Price Int, TaxrateInt, ***Primary Key(***Stockno)) ;

OUTPUT:

Table Structure is created in database:

Description of Table.

CONCLUSION:

Understand to design and develop relational database system by using MySql.

Assignment: 5

AIM:

Manage Data into the above tables using Insert, Select, Update, Delete DML SQL queries

PROBLEM STATEMENT /DEFINITION

Manage Data into the above tables using Insert, Select, Update, Delete with operators, functions, and set operator. And Execute queries like

- Display all the Purchase orders of a specific Customer.
- Get Customer and Data Item Information for a Specific Purchase Order.
- Get the Total Value of Purchase Orders.
- List the Purchase Orders in descending order as per total.
- Display the name of customers whose first name starts with “Rav”. (String matching :Like operator)
- Display the name of customer whose order amount is greater than all the customers. (Relational Operator: <, >, <=, >=, = =,!=)
- Display order details of customer whose city name is “Pune” and purchase date is “22/08/2016” (Boolean Operators: and, or)
- Add discount of 5% to all the customers whose order is more than Rs. 10000/-. (Arithmetic Operators +, -, *, /)
- Delete Purchase Order 1001.

OBJECTIVE:

1. To understand use of various DML quieries.
2. To understand use of various clauses: operators, functions, and set operator in DML queries.

THEORY:

1.3 DATA MANIPULATION LANGUAGE (DML):

After the database structure is defined with DDL, database administrators and users can utilize the Data Manipulation Language to insert, retrieve and modify the data contained within it.

a. INSERT COMMAND:

The INSERT command in SQL is used to add records to an existing table.

Format 1:-Inserting a single row of data into a table

Syntax
INSERT INTO table name [(column name, column name)] VALUES (expression, expression);
To add a new employee to the personal_info table
Example
INSERT INTO customer values ('xeta',' lincon', ‘calcuuta’)
Format 2: Inserting data into a table from another table
Syntax
INSERT INTO table name SELECT column name, column name FROM table name.

b. UPDATE COMMAND:
The UPDATE command can be used to modify information contained within a table.
Syntax
UPDATE table nameSET column name=expression, column name=expression, WHERE column name=expression;
Each year, company gives all employees a 3% cost-of-living increase in their salary. The following SQL command could be used to quickly apply this to all of the employees stored in the database:
Example
UPDATE account SET balance=balance*1.03

c. DELETE COMMAND:
The DELETE command can be used to delete information contained within a table.
Syntax
DELETE FROM table name WHERE search condition
The DELETE command with a WHERE clause can be used to remove his record from the personal info table:
Example
DELETE FROM account WHERE account-number=12345
The following command deletes all the rows from the table

Example:
DELETE FROM account;

INPUT: Given Database Schema
OUTPUT: Database tables for the given input

REFERENCE BOOK:

I. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw HillPublishers, ISBN 0-07-120413-X
II. Complete Reference of MySqlMcGraw Hill Publishers

Sample Questions:

Question: 1

Book = { Book_No ,Book_Name, Author_name , Cost, Category}

Member = { M_Id , M_Name ,Mship_type, Fees_paid,Max_Books_Allowed, Penalty_Amount }

Issue ={Lib_Issue_Id , Book_No , M_Id, Issue_Date, Return_date}

- List top 5 books which are issued by Annual members
- List the names of members who has issued the books whose cost is more than 300 rupees and whose author is "Scott Urman"
- Write a query to display number of booked in each category of books issued by all member types.

Question: 2

Consider the relational database :

dept (dept-no, dname, LOC)
emp (emp-no, ename, designation,sal)
project (proj-no, proj-name, status)

dept and emp are related as 1 to many.
project and emp are related as 1 to many.
Write relational or sq 1 expressions for the following :

i) List all employees of 'INVENTORY' department of 'PUNE' location.
ii) Give the names of employees who are working on 'Blood Bank' project.
iii) Give the name of managers from 'MARKETING' department.
iv) Give all the employees working under status 'INCOMPLETE' projects.
v)Write a Function that take Employee Number and return all the information related to the employee working on the project.
vi) Write a Procedure block that updates the salaries of the employees as per the following rules.

- If the designation is CLERK and deptno is 10 then increase the salary by 20%
- If the designation is MANAGER and deptno is 20 then increase the salary by 5%
- For all the other cases increase the salary by 10%

Question: 3

A database consists of following tables.

PROJECT(PNO, PNAME, CHIEF)

EMPLOYEE(EMPNO, EMPNAME)

ASSIGNED(PNO,EMPNO)

A. Get count of employees working on project.

B. Get details of employee working on project pr002.

C. Get details of employee working on project DBMS.

D. Write a trigger to delete all corresponding records from assigned table if employee id deleted.

E. Write a trigger to keep back up of assign table records if project is deleted.

Question: 4

Employee = (emp_no, emp_name, hiredate, comm., netsal ,dept_no, , designation)

1. Display all the employee details in department 30.
2. List the names, numbers and departments of all clerks.
3. Find the employees whose commission is greater than their salaries.
4. Find the employees whose commission is greater than 60% of their salaries.
5. List the name job and salary of all the employees in department 20 who earn more than 2000/-.
6. Find all the clerks in department 30 whose salary is greater than 1500/-.
7. Find all employees whose designation is either manager or president.
8. Find all managers who are not in department 30.
9. Find all the details of all the managers and clerks in department 10.
10. Find the details of the managers in department 10 and all clerks in department 20.

Question: 5

Employees (Employee_id,first_name , last_name , email, ph_no , hire_date, job_id, Salary, department_id)

Works (Employee_id,manager_id)

Departments (Department_id,dept_name , location_id)

Jobs (Job_id, job_title ,min_salary , max_salary)

Locations (Location_id , street, city, state , country)

Job_history(Employee_id , hire_date, leaving_date, salary, job_id, department_id)

1] Display all the employees in descending order of their salary.

2] Display employee_id, full name and salary of all employees who have joined in year 2006 according to their seniority.

3] List name of all departments in location 20,30 and 50

4] Display the full name of all employees whose first_name or last_name contains 'a'

5] Find the department with the most employees.

6] Display the department id and the count of the total no. of employees in respective departments in descending order by department id if count is > 5

7] Find those departments whose employees earn a higher salary, on average, than the average salary at department id 30.

8] List previous details of all employees who changed their department.

9] Display the manager_id and salary of the lowest paid employee for that manager. Exclude anyone whose manager is not known. Exclude any group where the minimum salary is less than $5000. Sort the output in descending order of salary.

10] **Write a procedure that accepts deptno value from a user, selects the maximum salary and minimum salary paid in the department, from the EMP table**

Question: 6

Create following tables in oracle

Emp(eno, ename, sal, contact_no, addr, dno)

project(pno, pname)

dept(dno, dname, loc)

assigned_to(eno, pno)

Write the SQL queries:

1. Gather details of employees working on project 353 and 354.
2. Obtains the details of employees working on the database project 107.
3. Find the employee nos of employees who work on at least one project that employee 107 works on.
4. Find the employee no of employees who work on all of the projects that employee 107 works on.
5. Find the project with minimum no of employees.
6. Create view to store pno, pname and no of employees working on the project.
7. Write a procedure to display details of the employees working on particular project. Use cursor.
8. Write a function to count no of employees working on particular project without using aggregate function.

Question: 7

instructor (ID ,name , dept_name, salary)

student (ID, name, dept_name , tot_cred)

takes (ID ,course_id , sec_id, semester,year , grade)

course(course_id,title , dept_name , credits)

classroom (building,room_number,capacity)

advisor(s_id,i_id)

Prereq(course_id,prereq_id)

Department(dept_name,building,budget)

Section(course_id,sec_id,semester,year,building,room_number,time_slot_id)

Teaches(id,course_id,sec_id,semester,year,grade)

Time_slot(time_slot_id,day,start_time,end_time)

1. Find the number of instructors who have never taught any course.
2. Find the total capacity of every building in the university
3. Find the maximum number of teachers for any single course section
4. Find all departments that have at least one instructor, and list the names of the departments along with the number of instructors; order the result in descending order of number of instructors.

5. As in the previous question, but this time you shouold include departments even if they do not have any instructor, with the count as 0
6. For each student, compute the total credits they have successfully completed, i.e. total credits of courses they have taken, for which they have a non-null grade other than 'F'. Do NOT use the tot_creds attribute of student.
7. Find the number of students who have been taught (at any time) by an instructor named 'Srinivasan'. Make sure you count a student only once even if the student has taken more than one course from Srinivasan.

Question 8

You need to create a movie database.

Create three tables, one for

actors(AID, name),

movies(MID, title) and

actor_role(MID, AID, rolename).

Use appropriate data types for each of the attributes, and add appropriate primary/foreign key constraints.

1. Insert data to the above tables (approx 3 to 6 rows in each table), including data for actor "Charlie Chaplin", and for yourself (using your roll number as ID).
2. Write a query to list all movies in which actor "Charlie Chaplin" has acted, along with the number of roles he had in that movie.
3. Write a query to list all actors who have not acted in any movie
4. List names of actors, along with titles of movies they have acted in. If they have not acted in any movie, show the movie title as null. (Do not use SQL outerjoin syntax here, write it from scratch.)

Question: 9

Consider the relational database

Supplier (sid, sname, address)

Parts (pid, pname, color)

Catalog (sid, pid, cost)

Write SQL queries for the following:

i) Find names of suppliers who supply some red parts.

ii) Find names of all parts whose cost is more than Rs. 25

iii) Find name of all parts whose color is green.

iv) Find name of supplier and parts with its color and cost.

Question: 10

Consider the relational database

Employee (person-name, street, city)

works (person-name, company-name, salary)

Company (company-name, city)

Manages (person-name, manager-name)

Consider the above relational database.

Write SQL queries for the following:

1. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annum.
2. Find the names of all employees in this database who live in the same city as the company for which they work.
3. Find the names of all employees who live in the same city and on the same street as do their managers.
4. Write a Trigger on update of employee company_name

Question: 11

Consider following database:

Student (Roll_no, Name, Address)

Subject (Sub_code, Sub_name)

Marks (Roll_no, Sub_code, marks)

Write following queries in SQL:

i) Find average marks of each student, along with the name of student.

ii) Find how many students have failed in the subject “DBMS”.

iii) Write a Trigger that check the rollno must be start with ‘TE’.

REFERENCE BOOK:

I. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw HillPublishers, ISBN 0-07-120413-X
II. Complete Reference of MySqlMcGraw Hill Publishers

CONCLUSION:

1. To study various basic DML commands.
2. To study various clauses written with DML commands like operators, functions, and set operator.

Assignment: 6

AIM:

Write various conditional select queries (DML) on above DB.

PROBLEM STATEMENT /DEFINITION

Write following conditional select queries on above DB.

A]. Aggregate functions (count, sum, avgetc)

- Get the total no of customers.
- Display average purchase amount of all the customers.
- Display total purchase amount of all the customers.

B]. Built in functions (now (), date (), day (), time () etc)

- Find DAYNAME, MONTHNAME and YEAR of the purchase order made on "1995-11-2016"
- Get current date & time, current time, current date
- Get 6 month future & past date using interval function based on current date and name the column accordingly.
- Find purchase details of the customers group by product category.

Find the purchase details of all the customers who made shopping today.(Using having clause)

OBJECTIVE:

1. To understand use of various DML, and DQL statement with clause and nested queries, DCL commands.
2. To understand use of various functions: Aggregate, Arithmetic and nested queries.
3. To understand the SQL Built in functions (now (), date (), day (), time () etc)

THEORY:

SELECT COMMAND:

The SELECT command is the most commonly used command in SQL. It allows database users to retrieve the specific information they desire from an operational database. Syntax

SELECT A1, A2…….. FROM table name WHERE predicate

A1, A2 is the list of attributes and predicate is the condition which must be satisfied by the resulting rows.

Example 1

It displays list of all last names in personal_info table

SELECT last_name FROM personal_info

The command shown below retrieves all of the information contained within the personal_info table. The asterisk is used as a wildcard in SQL. This means "Select everything from the personal_info table."

Example 2

SELECT * FROM account

Finally, the WHERE clause can be used to limit the records that are retrieved to those that meet specified criteria. The CEO might be interested in reviewing the personnel records of all highly paid employees. The following command retrieves all of the data contained within personal_info for records that have a salary value greater than $50,000:

Example 3

SELECT * FROM account WHERE balance > $50000

INPUT: A specific query set.

OUTPUT: Result of the given query set.

AGGREGATE FUNCTIONS:

If we want to summarize our data to produce top-level reports (For example, the purchasing manager may not be interested in a listing of all widget sales, but may simply want toknow thenumber of widgets sold this month),

SQL provides aggregate functions to assist with the summarization of large volumes of data.

OrderIDFirstName LastName Quantity UnitPrice Continent

122	John	Jacob	21	4.52	North America
923	Ralph	Wiggum	192	3.99	North America
238	Ryan	Johnson	87	4.49	Africa
829	Mary	Smith	842	2.99	North America
824	Elizabeth	Marks	48	3.48	Africa
753	James	Linea	9	7.85	North America
942	Alan	Jonas	638	3.29	Europe

1. SUM:

It is used within a SELECT statement and, predictably, returns the summation of a series of values. If the widget project manager wanted to know the total number of widgets sold to date, we could use the Following query:

Example:

SELECT SUM (salary) AS TOTAL FROM account

Total

1837

2. AVG:

The AVG (average) function works in a similar manner to provide the mathematical average of a series of values. To find out the average amount of all orders placed on the North American continent.

Example:
SELECT AVG (balance) as AveragePrice FROM account WHERE branch-name='delhi'

3. COUNT:

SQL provides the COUNT function to retrieve the number of records in a table that meet given criteria. We can use the COUNT (*) syntax alone to retrieve the number of rows in a table. Alternatively, a WHERE clause can be included to restrict the counting to specific records.

Example:
SELECT COUNT(*) AS 'Number of Large Orders' FROM account WHERE BALANCE > 100000 Above query displays how many orders are processed by company that requested over 100 widgets.

4. MAX AND MIN:

SQL provides Min and Max functions to locate the records containing the smallest and largest values for a given expression

The MAX () function returns the largest value in a given data series. We can provide the function with a field name to return the largest value for a given field in a table. To find the order in our database that produced the most revenue for the company, following query will be used:

Example:

SELECT MAX (BALANCE) as 'largest balance' FROM account

The MIN () function functions in the same manner, but returns the minimum value for the expression.

B) SQL DATE DATA TYPES:

MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

SQL Server comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Note: The date types are chosen for a column when you create a new table in your database!

SQL Working with Dates

You can compare two dates easily if there is no time component involved!

Assume we have the following "Orders" table:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Now we want to select the records with an OrderDate of "2008-11-11" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

The result-set will look like this:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

Now, assume that the "Orders" table looks like this (notice the time component in the "OrderDate" column):

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

If we use the same SELECT statement as above:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

we will get no result! This is because the query is looking only for dates with no time portion.

Tip: To keep your queries simple and easy to maintain, do not allow time components in your dates!

REFERENCE BOOK:

1. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw Hill Publishers, ISBN 0-07-120413-X.
2. The complete Reference Mysql-McGraw Hill.
3. DBMS Complete Practical Approach-Maheshwari,Jain

CONCLUSION:

Understand to retrieve data from database with the help of sql statement and operators.

Assignment: 7

AIM: Write following nested sub queries on above DB.

PROBLEM STATEMENT /DEFINITION

Write following nested sub queries on above DB.

A]. set membership(in, not in)

- Get order details of products which are not from electronics and sports category.
- Get the name and quantity of product which have either 10 or 20 or 30 quantities.

B]. set comparison (<,>,<=,>=, <some, >=some, <all etc.)

- Get the product details whose product price is more than “Apple 7”.
- Find the purchase order whose purchase amount is greater than maximum purchase amount.

Also use following keywords in nested sub queries.

EXISTS /NOT EXISTS, ANY etc.

OBJECTIVE

- To understand nested sub-queries.
  - set membership(in, not in)
  - set comparison (<,>,<=,>=, <some, >=some, <all etc.)

THEORY:

Types of Join :

a) INNER JOIN:
The INNER JOIN returns all rows from both tables where there is a match. If there are rows in Employees that do not have matches in Orders, those rows will not be listed.

Syntax:
SELECT field1, field2, field3 FROM first_table INNER JOIN second_table ON first_table.keyfield = second_table.foreign_keyfield
Example:
Who has ordered a product, and what did they order?

SELECT Employees.Name, Orders.Product FROM Employees INNER JOIN Orders ON Employees.Employee_ID=Orders.Employee_ID

Result

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

b) LEFT OUTER JOIN:

The LEFT JOIN returns all the rows from the first table (Employees), even if there are no matches in the Second table (Orders). If there are rows in Employees that do not have matches in Orders, those rows also will be listed.

Syntax:

SELECT field1, field2, field3 FROM first_table LEFT JOIN second_table ON first_table.keyfield = second_table.foreign_keyfield

Example:

List all employees, and their orders - if any.

SELECT Employees.Name, Orders.Product FROM Employees LEFT JOIN OrdersON Employees.Employee_ID=Orders.Employee_ID

Result:

Name	Product
Hansen, Ola	Printer
Svendson, Tove	
Svendson, Stephen	Table
Svendson, Stephen	Chair
Pettersen, Kari	

c) RIGHT OUTER JOIN:

The RIGHT JOIN returns all the rows from the second table (Orders), even if there are no matches in the first table (Employees). If there had been any rows in Orders that did not have matches in Employees, those rows also would have been listed.

Syntax:

SELECT field1, field2, field3 FROM first_table RIGHT JOIN second_table ON first_table.keyfield = second_table.foreign_keyfield

Example:
List all orders, and who has ordered - if any.

SELECT Employees.Name, Orders.Product FROM Employees RIGHT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID

Result

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Execute queries like:

1. Display all the Purchase orders of a specific Customer.

*SELECT * FROM Customer C ,PurchaseOrder P Where C.CustNo = P.CustNo AND C.CustName = 'XXXXX' ;*

2.Get Customer and Data Item Information for a Specific Purchase Order.

SELECT C.CustNo, C.CustName, C.Street, C.City, C.State, C.Zip,
P.PONo, P.OrderDate,CO.StockNo, CO.Quantity, CO.Discount
FROM Customer C, PurchaseOrder P, Contains CO
WHERE C.CustNo = P.CustNo
AND P.PONo = CO.PONo
AND P.PONo = 1001 ;

3. Get the Total Value of Purchase Orders.

*SELECT P.PONo, SUM(S.Price * CO.Quantity)SELECT P.PONo, SUM(S.Price * CO.Quantity)FROM PurchaseOrder P, Contains CO, Stock S*
WHERE P.PONo = CO.PONoAND CO.StockNo = S.StockNoGROUP BY P.PONo ;

4. List the Purchase Orders in descending order as per total.

*CREATE VIEW X(Purchase,Total) ASSELECT P.PONo, SUM(S.Price * CO.Quantity)*
FROM PurchaseOrder P, Contains CO, Stock SWHERE P.PONo = CO.PONo
AND CO.StockNo = S.StockNoGROUP BY P.PONO

*SELECT *FROM XORDER BY Total desc ;*

The SQL IN Operator:

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

or:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

IN Operator Examples

The following SQL statement selects all customers that are located in "Germany", "France" and "UK":

Example

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

C) set comparison (<,>,<=,>=, <some, >=some, <all etc.)

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

REFERENCE BOOK:

1. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw Hill Publishers, ISBN 0-07-120413-X.
2. The complete Reference Mysql-McGraw Hill.
3. DBMS Complete Practical Approach-Maheshwari,Jain

CONCLUSION:

- Understand to retrieve data from various database tables with the help of nested sub-queries.

Assignment: 8

AIM:

Write and execute PL/SQL block to implement all types of triggers on above DB.

(Consider row level and statement level triggers)

PROBLEM STATEMENT /DEFINITION

Write trigger and execute it on a table.

Write Simple PL/SQL programs to perform different operations on tables

OBJECTIVE:

To understand PL/SQL
To understand the concept of **triggers, pl /sql block**.

THEORY:

1.TRIGGER ON UPDATE DISPLAY ERROR MESSAGE:

```
CREATE TRIGGER account_bal

BEFORE UPDATE

ON account

FOR EACH ROW

BEGIN

IF (NEW.bal< 0) THEN

SIGNAL SQLSTATE '80000'

SET MESSAGE_TEXT='Account balance cannot be less than 0';

END IF;

END;
```

1. Create trigger on update which will create log file of update

```
CREATE TRIGGER account_bal1

AFTER UPDATE ON account

 FOR EACH ROW

BEGIN

INSERT into transaction_log(user_id, description)
```

```
VALUES (user( ),CONCAT('Adjusted account ',NEW.accno,' from ',OLD.bal,' to ', NEW.bal));

end;
```

Sample PL/SQL Statement:

1. Write a PL/SQL block to find the sum of first 100 odd nos. and even nos)

2. Write a PL/SQL block to display the Information of given student on following table Stud (sno, sname, address, city).

3. Write a PL/SQL block for preparing a Net Salary, given employee on following table
   Emp (eno, ename, address, city)
   Salary (eno, basic, da, hra, it)
   Net_Salary (eno, total_allowance, total_deduction, netpay)
   Notes : D.A. = 59% of basic , H.R.A. = 500, I.T. = 2% of basic
   Total_Allowance = Basic + D.A. + H.R.A., Total_Deduction = I.T.
   Netpay = Total_Allowance – Total_Deduction.

4. Write a PL/SQL block to raise the salary by 20% of given employee on following table.

   Emp_Salary (eno, ename, city, salary)

5. Write an Implicit Cursor to accept the employee number from the user. You have to delete this record and display the appropriate message on the following table.
   Emp (eno, ename, address, city)

6. Write a Cursor to display the employee number, name, department and salary of first employee getting the highest salary.

   Emp (eno, ename, department, address, city)
   Salary (eno, salary)

7. Write a Cursor to display the first five records on the following table.

   Student(sno, sname, address, city)

8. Write Cursor for preparing a Net Salary for employee's of finance department and Net Pay is more than 10,000 on following table.
9. Writes a Function to check whether the given number is prime or not

REFERENCE BOOK:

4. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw Hill Publishers, ISBN 0-07-120413-X.
5. The complete Reference Mysql-McGraw Hill.
6. DBMS Complete Practical Approach-Maheshwari,Jain

CONCLUSION:

- Understand the concept of PL/SQL block by implementing all types of triggers on DB

Assignment: 9

AIM:

Write and execute PL/SQL stored procedure and function to perform a suitable task on above DB.

PROBLEM STATEMENT /DEFINITION

1. PL/SQL Assignmentsbased on tables created.
2. Write Simple PL/SQL programs to perform different operations on tables
3. Write trigger and execute it on a table.

OBJECTIVE

- To understand PL/SQL
- To understand the concept of **procedures /Functions**

THEORY:

A) Creating a Stored Procedure:

Syntax to create Stored Procedure in mySql:

```
DELIMITER //

CREATE PROCEDURE p2()

LANGUAGE SQL

DETERMINISTIC

SQL SECURITY DEFINER

COMMENT 'A procedure'

BEGIN

   SELECT 'Hello World !';

END//
```

The first part of the statement creates the procedure. The next clauses defines the optional characteristics of the procedure. Then you have the name and finally the body or routine code.

Stored procedure names are case insensitive, and you cannot create procedures with the same name. Inside a procedure body, you can't put database-manipulation statements.Modify a Stored Procedure

MySQL provides an ALTER PROCEDURE statement to modify a routine, but only allows for the ability to change certain characteristics. If you need to alter the body or the parameters, you must drop and recreate the procedure.

Delete a Stored Procedure

DROP PROCEDURE IF EXISTS p2;

CREATE PROCEDURE proc1 (IN varname DATA-TYPE) : One input parameter. The word IN is optional because parameters are IN (input) by default.

CREATE PROCEDURE proc1 (OUT varname DATA-TYPE) : One output parameter.

CREATE PROCEDURE proc1 (INOUT varname DATA-TYPE) : One parameter which is both input and output.

Of course, you can define multiple parameters defined with different types.

IN example

```
DELIMITER //

CREATE PROCEDURE proc_IN (IN var1 INT)

BEGIN

   SELECT var1 + 2 AS result;

END//
```

OUT example

```
DELIMITER //

CREATE PROCEDURE proc_OUT (OUT var1 VARCHAR(100))

BEGIN

   SET var1 = 'This is a test';

END //
```

INOUT example

```
DELIMITER //

CREATE PROCEDURE proc_INOUT(INOUT var1 INT)
```

```
BEGIN
   SET var1 = var1 * 2;
END //
```

Step 4 - Variables

The following step will teach you how to define variables, and store values inside a procedure. You must declare them explicitly at the start of the BEGIN/END block, along with their data types. Once you've declared a variable, you can use it anywhere that you could use a session variable, or literal, or column name.

Declare a variable using the following syntax:

```
DECLARE varname DATA-TYPE DEFAULT defaultvalue;
```

Let's declare a few variables:

```
DECLARE a, b INT DEFAULT 5;
DECLARE strVARCHAR(50);
DECLARE today TIMESTAMP DEFAULT CURRENT_DATE;
DECLARE v1, v2, v3 TINYINT;
```

Working with variables

Once the variables have been declared, you can assign them values using the SET or SELECT command:

```
DELIMITER //
CREATE PROCEDURE var_proc(IN paramstr VARCHAR(20))
BEGIN
   DECLARE a, b INT DEFAULT 5;
   DECLARE strVARCHAR(50);
   DECLARE today TIMESTAMP DEFAULT CURRENT_DATE;
   DECLARE v1, v2, v3 TINYINT;
   INSERT INTO table1 VALUES (a);
```

```
    SET str = 'I am a string';

    SELECT CONCAT(str,paramstr), today FROM table2 WHERE b >=5;

END //
```

Step 5 - Flow Control Structures

MySQL supports the IF, CASE, ITERATE, LEAVE LOOP, WHILE and REPEAT constructs for flow control within stored programs. We're going to review how to use IF, CASE and WHILE specifically, since they happen to be the most commonly used statements in routines.

- **IF statement**

With the IF statement, we can handle tasks which involves conditions:

```
DELIMITER //

CREATE PROCEDURE proc_IF(IN param1 INT)

BEGIN

    DECLARE variable1 INT;

    SET variable1 = param1 + 1;

     IF variable1 = 0 THEN

        SELECT variable1;

    END IF;

    IF param1 = 0 THEN

        SELECT 'Parameter value = 0';

    ELSE

        SELECT 'Parameter value <> 0';

    END IF;

END //
```

- **CASE statement**

The CASE statement is another way to check conditions and take the appropriate path. It's an excellent way to replace multiple IF statements. The statement can be written in two different ways, providing great flexibility to handle multiple conditions.

```
DELIMITER //
CREATE PROCEDURE proc_CASE(IN param1 INT)
BEGIN
   DECLARE variable1 INT;
   SET variable1 = param1 + 1;
  CASE variable1
      WHEN 0 THEN
         INSERT INTO table1 VALUES (param1);
      WHEN 1 THEN
         INSERT INTO table1 VALUES (variable1);
      ELSE
         INSERT INTO table1 VALUES (99);
   END CASE;
END //
```

or:

```
DELIMITER //
CREATE PROCEDURE proc_CASE (IN param1 INT)
BEGIN
   DECLARE variable1 INT;
   SET variable1 = param1 + 1;
      CASE
      WHEN variable1 = 0 THEN
```

```
        INSERT INTO table1 VALUES (param1);
    WHEN variable1 = 1 THEN
        INSERT INTO table1 VALUES (variable1);
    ELSE
        INSERT INTO table1 VALUES (99);
  END CASE;
END //
```

- **WHILE statement**

There are technically three standard loops: WHILE loops, LOOP loops, and REPEAT loops. You also have the option of creating a loop using the "Darth Vader" of programming techniques: the GOTO statement. Check out this example of a loop in action:

```
DELIMITER //
CREATE PROCEDURE proc_WHILE (IN param1 INT)
BEGIN
  DECLARE variable1, variable2 INT;
  SET variable1 = 0;
    WHILE variable1 < param1 DO
    INSERT INTO table1 VALUES (param1);
    SELECT COUNT(*) INTO variable2 FROM table1;
    SET variable1 = variable1 + 1;
  END WHILE;
END //
```

REFERENCE BOOK:

7. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw Hill Publishers, ISBN 0-07-120413-X.

8. The complete Reference Mysql-McGraw Hill.
9. DBMS Complete Practical Approach-Maheshwari,Jain

CONCLUSION:

- Understand the concept of PL/SQL block by implementing all types procedure and function on DB.

Assignment: 10

AIM:

Write and execute PL/SQL block to implement all types of cursor on above DB.

PROBLEM STATEMENT /DEFINITION

4. PL/SQL Assignments based on tables created.
5. Write Simple PL/SQL programs to perform different operations on tables
6. Write cursor and execute it on a table.

OBJECTIVE

- To understand PL/SQL
- To understand the concept of **cursor**

THEORY:

Cursor:

Oracle creates a memory area, known as the context area, for processing an SQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to

be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the**SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK_ROWCOUNT** and**%BULK_EXCEPTIONS**, designed for use with the **FORALL**statement. The following table provides the description of the most used attributes −

S.No	Attribute & Description
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as**sql%attribute_name** as shown below in the example.

Example

We will be using the CUSTOMERS table we had created and used in the previous chapters.

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program will update the table and increase the salary of each customer by 500 and use the **SQL%ROWCOUNT**attribute to determine the number of rows affected −

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
6 customers selected
```

```
PL/SQL procedure successfully completed.
```

If you check the records in customers table, you will find that the rows have been updated −

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2500.00 |
|  2 | Khilan   |  25 | Delhi     |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2500.00 |
|  4 | Chaitali |  25 | Mumbai    |  7000.00 |
|  5 | Hardik   |  27 | Bhopal    |  9000.00 |
|  6 | Komal    |  22 | MP        |  5000.00 |
+----+----------+-----+-----------+----------+
```

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
```

```
SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
   c_id customers.id%type;
   c_name customerS.No.ame%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
```

```
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP

PL/SQL procedure successfully completed.
```

REFERENCE BOOK:

1. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw Hill Publishers, ISBN 0-07-120413-X.
2. The complete Reference Mysql-McGraw Hill.
3. DBMS Complete Practical Approach-Maheshwari,Jain

CONCLUSION:

- Understand the concept of PL/SQL block by implementing all types cursor on DB.

Assignment: 11

AIM:

Write DDL statements to create VIEWS on single and multiple tables from above DB.

PROBLEM STATEMENT /DEFINITION

Write DDL statements to create VIEWS on single and multiple tables from above DB.

Do the following operation to demonstrate the use of view:

- Update the base table
- Insert new record in the base table.
- Delete record in the base table.
- DML on VIEW.

What are the restrictions applicable while creating or modifying views? Demonstrate using suitable queries.

OBJECTIVE

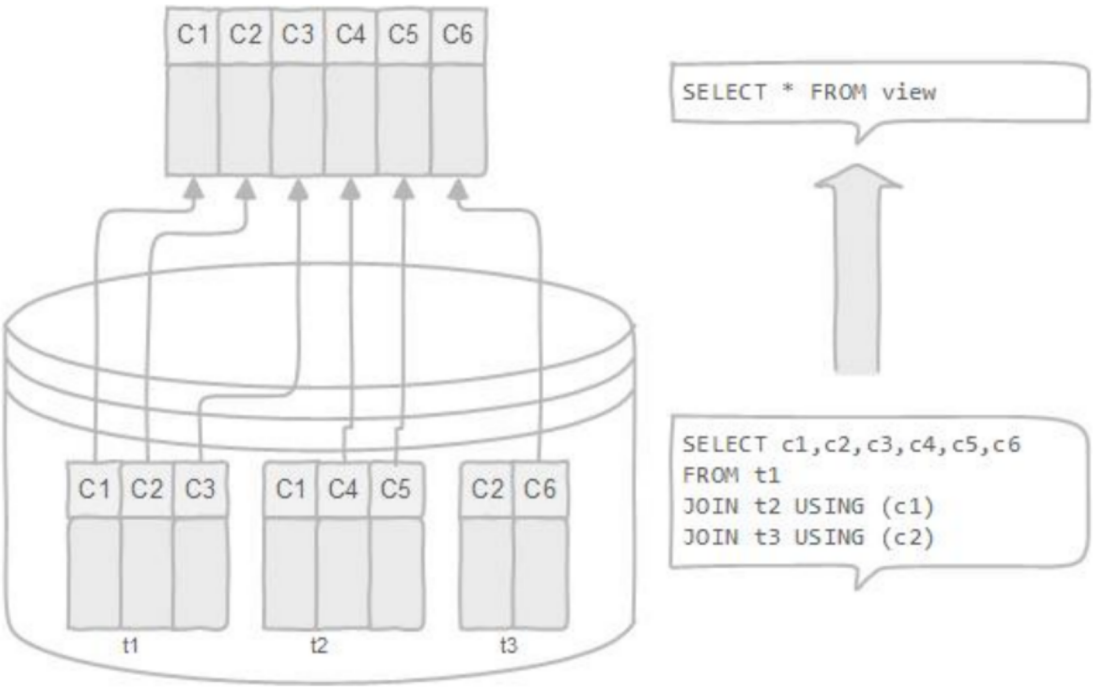
1. To understand use of VIEWS.
2. To understand implementation of different types of VIEWS.
3. To implement and analyze various operations on VIEWS.

THEORY:

VIEW:

Database view is known as a "virtual table" that allows you to query the data in it. Understanding the database views and using them correctly are very important. In this section, we will discuss the database views, how they are implemented in MySQL, and how to use them more effectively.

A database view is a virtual table or logical table which is defined as a SQL SELECT query with joins. Because a database view is similar to a database table, which consists of rows and columns, so you can query data against it. Most database management systems, including MySQL, allow you to update data in the underlying tables through the database view with some prerequisites.



A database view is dynamic because it is not related to the physical schema. The database system stores database views as a SQL SELECT statement with joins. When the data of the tables changes, the view reflects that changes as well.

Advantages of database view

The following are advantages of using database views.

- A database view allows you to simplify complex queries: a database view is defined by an SQL statement that associates with many underlying tables. You can use database view to hide the complexity of underlying tables to the end-users and external applications. Through a database view, you only have to use simple SQL statements instead of complex ones with many joins.
- A database view helps limit data access to specific users. You may not want a subset of sensitive data can be queryable by all users. You can use a database view to expose only non-sensitive data to a specific group of users.
- A database view provides extra security layer. Security is a vital part of any relational database management system. The database view provides extra security for a

database management system. The database view allows you to create the read-only view to expose read-only data to specific users. Users can only retrieve data in read-only view but cannot update it.

- A database view enables computed columns. A database table should not have calculated columns however a database view should. Suppose in the`orderDetails` table you have `quantityOrder` (the number of ordered products) and `priceEach` (price per product item) columns. However, the `orderDetails` table does not have a computed column to store total sales for each line item of the order. If it has, the database schema would not be a good design. In this case, you can create a computed column named `total` , which is a product of`quantityOrder` and `priceEach` to represent the computed result. When you query data from the database view, the data of the computed column is calculated on fly.
- A database view enables backward compatibility. Suppose you have a central database, which many applications are using it. One day, you decide to redesign the database to adapt with the new business requirements. You remove some tables and create new tables, and you don't want the changes affect other applications. In this scenario, you can create database views with the same schema as the legacy tables that you will remove.

Disadvantages of database view

Besides the advantages above, there are several disadvantages of using database views:

- Performance: querying data from a database view can be slow especially if the view is created based on other views.
- Tables dependency: you create a view based on underlying tables of the a database. Whenever you change the structure of those tables that view associated with, you have to change the view as well.

In this tutorial, you have learned what a database view is. We also discussed the advantages and disadvantages of using database views so that you can apply them effectively in your database design.

Introduction to CREATE VIEW statement

To create a new view in MySQL, you use the `CREATE VIEW` statement. The syntax of creating a view in MySQL is as follows:

```
1 CREATE
2    [ALGORITHM = {MERGE  | TEMPTABLE | UNDEFINED}]
3 VIEW [database_name].[view_name]
4 AS
5 [SELECT  statement]
```

Let's examine the syntax in more detail.

View processing algorithms

The algorithm attribute allows you to control which mechanism MySQL uses when creating the view. MySQL provides three algorithms: `MERGE`, `TEMPTABLE`, and`UNDEFINED`.

- Using `MERGE` algorithm, MySQL first combines the input query with the SELECT statement, which defines the view, into a single query. And then MySQL executes the combined query to return the result set. The `MERGE` algorithm is not allowed if the `SELECT` statement contains aggregate functions such as MIN, MAX, SUM,COUNT, AVG or DISTINCT, GROUP BY, HAVING, LIMIT, UNION, UNION ALL,subquery. In case the `SELECT` statement refers to no table, the `MERGE` algorithm is also not allowed. If the `MERGE` algorithm is not allowed, MySQL changes the algorithm to `UNDEFINED`. Note that the combination of input query and the query in the view definition into one query is referred to as *view resolution*.
- Using `TEMPTABLE` algorithm, MySQL first creates a temporary table based on the`SELECT` statement that defines the view, and then it executes the input query against this temporary table. Because MySQL has to create a temporary table to store the result set and moves the data from the base tables to the temporary table, the `TEMPTABLE` algorithm is less efficient than the `MERGE` algorithm. In addition, a view that uses `TEMPTABLE` algorithm is not updatable.
- The `UNDEFINED` is the default algorithm when you create a view without specifying an explicit algorithm. The `UNDEFINED` algorithm lets MySQL make a choice of using `MERGE` or `TEMPTABLE` algorithm. MySQL

prefers `MERGE` algorithm to`TEMPTABLE` algorithm because the `MERGE` algorithm is much more efficient.

View name

Within a database, views and tables share the same namespace, therefore, a view and a table cannot have the same name. In addition, the name of a view must follow the table's naming rules.

SELECT statement

In the `SELECT` statement, you can query data from any table or view that exists in the database. There are several rules that the `SELECT` statement must follow:

- The `SELECT` statement can contain a subquery in WHERE clause but not in the`FROM` clause.
- The `SELECT` statement cannot refer to any variables including local variables, user variables, and session variables.
- The `SELECT` statement cannot refer to the parameters of prepared statements.

Note that the `SELECT` statement needs not to refer to any tables.

Creating view in MySQL

Creating simple views

Let's take a look at the `orderDetails` table. We can create a view that represents total sales per order.

```
1 CREATE VIEW SalePerOrder AS
2    SELECT
3        orderNumber, SUM(quantityOrdered * priceEach) total
4    FROM
5        orderDetails
6    GROUP by orderNumber
7    ORDER BY total DESC;
```

If you use the `SHOW TABLE` command to view all tables in the `classicmodels` database, we also see the `SalesPerOrder` view is showing up in the list.

```
1 SHOW TABLES;
```

	Tables_in_classicmodels
▶	customers
	employees
	offices
	orderdetails
	orders
	payments
	productlines
	products
	saleperorder

This is because the views and tables share the same namespace. To know which object is view or table, you use the `SHOW FULL TABLE` command as follows:

	Tables_in_classicmodels	Table_type
▶	customers	BASE TABLE
	employees	BASE TABLE
	offices	BASE TABLE
	orderdetails	BASE TABLE
	orders	BASE TABLE
	payments	BASE TABLE
	productlines	BASE TABLE
	products	BASE TABLE
	saleperorder	VIEW

The `table_type` column in the result set specifies which object is view and which object is a table (base table).
If we want to query total sales for each sales order, you just need to execute a simple`SELECT` statement against the `SalePerOrder` view as follows:

```
1 SELECT
2    *
3 FROM
4    salePerOrder;
```

	orderNumber	total
▶	10165	67392.84
	10287	61402
	10310	61234.66
	10212	59830.54
	10207	59265.14
	10127	58841.35
	10204	58793.53
	10126	57131.92

Creating a view based on another view

MySQL allows you to create a view based on another view. For example, you can create a view called big sales order based on the `SalesPerOrder` view to show every sales order whose total is greater than `60,000` as follows:

```
1 CREATE VIEW BigSalesOrder AS
2    SELECT
3        orderNumber, ROUND(total,2) as total
4    FROM
5        saleperorder
6    WHERE
7        total > 60000;
```

Now, we can query the data from the `BigSalesOrder` view as follows:

```
1 SELECT
2    orderNumber, total
3 FROM
4    BigSalesOrder;
```

	orderNumber	total
▶	10165	67392.85
	10287	61402.00
	10310	61234.67

Creating views with join

The following is an example of creating a view with INNER JOIN . The view contains the*order number*, *customer name,* and *total sales* per order.

```
1
2  CREATE VIEW customerOrders AS
3      SELECT
4          d.orderNumber,
5          customerName,
6          SUM(quantityOrdered * priceEach) total
7      FROM
8          orderDetails d
9              INNER JOIN
1          orders o ON o.orderNumber = d.orderNumber
0
               INNER JOIN
1
1          customers c ON c.customerNumber = c.customerNumber
1      GROUP BY d.orderNumber
2
       ORDER BY total DESC;
1
3
```

To query data from the `customerOrders` view, you use the following query:

```
1 SELECT
2     *
3 FROM
4     customerOrders;
```

	orderNumber	customerName	total
▶	10165	Dragon Souveniers, Ltd.	67392.84
	10287	Vida Sport, Ltd	61402
	10310	Toms Spezialitäten, Ltd	61234.66
	10212	Euro+ Shopping Channel	59830.54
	10207	Diecast Collectables	59265.14
	10127	Muscle Machine Inc	58841.35
	10204	Muscle Machine Inc	58793.53
	10126	Corrida Auto Replicas, Ltd	57131.92

Creating views with subquery

The following illustrates how to create a view with a subquery. The view contains products whose buy prices are higher than the average price of all products.

```
1
2  CREATE VIEW aboveAvgProducts AS
3    SELECT
4      productCode, productName, buyPrice
5    FROM
6      products
7    WHERE
8      buyPrice >
9  (SELECT
1            AVG(buyPrice)
0          FROM
1            products)
1
1    ORDER BY buyPrice DESC;
1
2
```

Querying data from the `aboveAvgProducts` is simple as follows:

```
1 SELECT
2   *
3 FROM
4   aboveAvgProducts;
```

	productCode	productName	buyPrice
▶	S10_4962	1962 LanciaA Delta 16V	103.42
	S18_2238	1998 Chrysler Plymouth Prowler	101.51
	S10_1949	1952 Alpine Renault 1300	98.58
	S24_3856	1956 Porsche 356A Coupe	98.3
	S12_1108	2001 Ferrari Enzo	95.59
	S12_1099	1968 Ford Mustang	95.34
	S18_1984	1995 Honda Civic	93.89
	S18_4027	1970 Triumph Spitfire	91.92

Showing view definition

MySQL provides the `SHOW CREATE VIEW` statement that displays the view's definition. The following is the syntax of the `SHOW CREATE VIEW` statement:

```
1 SHOW CREATE VIEW [database_name].[view_ name];
```

To display the definition of a view, you need to specify its name after the `SHOW CREATE VIEW` clause.

Let's create a view for the demonstration.

We create a simple view based on the `employees` table that displays the company's organization structure:

```
1 CREATE VIEW organization AS
2    SELECT
3       CONCAT(E.lastname, E.firstname) AS Employee,
4       CONCAT(M.lastname, M.firstname) AS Manager
5    FROM
6       employees AS E
7          INNER JOIN
8       employees AS M ON M.employeeNumber = E.ReportsTo
9    ORDER BY Manager;
```

	Employee	Manager
▶	BottLarry	BondurGerard
	JonesBarry	BondurGerard
	BondurLoui	BondurGerard
	GerardMartin	BondurGerard
	HernandezGerard	BondurGerard
	CastilloPamela	BondurGerard
	TsengFoon Yue	BowAnthony

To display the view's definition, you use the `SHOW CREATE VIEW` statement as follows:

```
1 SHOW CREATE VIEW organization;
```

You can also display the definition of the view using any plain text editor such as notepad to open the view definition file in the database folder.

For example, to open the `organization` view definition, you can find the view definition file with the following path: `\data\classicmodels\organization.frm`
However, you should not modify the view directly in the *.frm file.

Modifying views

MySQL provides two statements that allow you to modify an existing view: `ALTER VIEW`and `CREATE OR REPLACE VIEW`

Modifying views using ALTER VIEW statement

Once a view is created, you can modify it using the `ALTER VIEW` statement.
The syntax of the `ALTER VIEW` statement is similar to the `CREATE VIEW` statement except that the `CREATE` keyword is replaced by the `ALTER` keyword.

```
1 ALTER
2  [ALGORITHM =  {MERGE | TEMPTABLE | UNDEFINED}]
3   VIEW [database_name].  [view_name]
4    AS
5  [SELECT  statement]
```

The following statement modifies the `organization` view by adding the `email` column.

```
1 ALTER VIEW organization
2   AS
3   SELECT CONCAT(E.lastname,E.firstname) AS Employee,
4          E.email AS  employeeEmail,
5          CONCAT(M.lastname,M.firstname) AS Manager
6   FROM employees AS E
7   INNER JOIN employees AS M
8    ON M.employeeNumber = E.ReportsTo
9   ORDER BY Manager;
```

To verify the change, you can query data from the `organization` view:

```
1 SELECT
2    *
3 FROM
4    Organization;
```

	Employee	employeeEmail	Manager
▶	JonesBarry	bjones@classicmodelcars.com	BondurGerard
	HernandezGerard	ghernande@classicmodelcars.com	BondurGerard
	BottLarry	lbott@classicmodelcars.com	BondurGerard
	GerardMartin	mgerard@classicmodelcars.com	BondurGerard
	BondurLoui	lbondur@classicmodelcars.com	BondurGerard
	CastilloPamela	pcastillo@classicmodelcars.com	BondurGerard
	VanaufGeorge	gvanauf@classicmodelcars.com	BowAnthony

Modifying view using CREATE OR REPLACE VIEW statement

In addition to the ALTER VIEW statement, you can use CREATE OR REPLACE VIEW statement to either create or replace an existing view. If a view already exists, MySQL simply modifies the view. In case the view does not exist, MySQL creates a new view.

The following statement uses CREATE OR REPLACE VIEW syntax to create contacts view based on the employee table:

```
1 CREATE OR REPLACE VIEW contacts AS
2    SELECT
3        firstName, lastName, extension, email
4    FROM
5        employees;
```

	firstName	lastName	extension	email
▶	Diane	Murphy	x5800	dmurphy@classicmodelcars.com
	Mary	Patterson	x4611	mpatterso@classicmodelcars.com
	Jeff	Firrelli	x9273	jfirrelli@classicmodelcars.com
	William	Patterson	x4871	wpatterson@classicmodelcars.com
	Gerard	Bondur	x5408	gbondur@classicmodelcars.com
	Anthony	Bow	x5428	abow@classicmodelcars.com
	Leslie	Jennings	x3291	ljennings@classicmodelcars.com

Suppose you want to add the job title column to the contacts view, you just simply use the following statement.

```
1 CREATE OR REPLACE VIEW contacts AS
2    SELECT
3        firstName, lastName, extension, email, jobtitle
4    FROM
5        employees;
```

	firstName	lastName	extension	email	jobtitle
▶	Diane	Murphy	x5800	dmurphy@classicmodelcars.com	President
	Mary	Patterson	x4611	mpatterso@classicmodelcars.com	VP Sales
	Jeff	Firrelli	x9273	jfirrelli@classicmodelcars.com	VP Marketing
	William	Patterson	x4871	wpatterson@classicmodelcars.com	Sales Manager (APAC)
	Gerard	Bondur	x5408	gbondur@classicmodelcars.com	Sale Manager (EMEA)
	Anthony	Bow	x5428	abow@classicmodelcars.com	Sales Manager (NA)
	Leslie	Jennings	x3291	ljennings@classicmodelcars.com	Sales Rep

Removing views

Once a view created, you can remove it using the `DROP VIEW` statement. The following illustrates the syntax of the `DROP VIEW` statement:

```
DROP VIEW [IF EXISTS] [database_name].[view_name]
```

The `IF EXISTS` is the optional clause of the statement, which allows you to check whether the view exists or not. It helps you avoid an error of removing a non-existent view.

For example, if you want to remove the `organization` view, you can use the `DROP VIEW` statement as follows:

```
DROP VIEW IF EXISTS organization;
```

Each time you modify or remove a view, MySQL makes a backup of the view definition file to the `/database_name/arc/` folder. In case you modify or remove a view by accident, you can get its backup from the arc folder.

REFERENCE BOOK:

1. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", 6thEdition, McGraw Hill Publishers, ISBN 0-07-120413-X.
2. The complete Reference Mysql-McGraw Hill.
3. DBMS Complete Practical Approach-Maheshwari,Jain

CONCLUSION:

- Understand the concept of PL/SQL block by implementing all types cursor on DB.

GROUP C

MONGODB

Assignment: 12

AIM: Create a database with suitable example using MongoDB and implement basic commands on mongodb database.

PROBLEM STATEMENT /DEFINITION

Create a database with suitable example using MongoDB and implement

- Inserting and saving document (batch insert, insert validation)
- Removing document
- Updating document (document replacement, using modifiers, upserts, updating multiple documents, returning updated documents)

OBJECTIVE:

To understand Mongodb basic commands
To implement the concept of document oriented databases..

THEORY:

SQL VsMongoDB

SQL Concepts	MongoDB Concepts
Database	Database
Table	Collection
Row	Document 0r BSON Document
Column	Field
Index	Index
Table Join	Embedded Documents & Linking
Primary Key	Primary Key
Specify Any Unique Column Or Column Combination As Primary Key.	In Mongodb, The Primary Key Is Automatically Set To The _Id Field.
Aggregation (E.G. Group By)	Aggregation Pipeline

1.Create a collection in mongodb

db.createCollection(“Teacher_info")

2.Create a capped collection in mongodb

>db.createCollection("audit", {capped:true, size:20480})

{ "ok" : 1 }

3.Insert a document into collection

db.Teacher_info.insert({ Teacher_id: “Pic001", Teacher_Name: “Ravi",Dept_Name: “IT”, Sal:30000, status: "A" })

db.Teacher_info.insert({ Teacher_id: “Pic002", Teacher_Name: “Ravi",Dept_Name: “IT”, Sal:20000, status: "A" })

db.Teacher_info.insert({ Teacher_id: “Pic003", Teacher_Name: “Akshay",Dept_Name: “Comp”, Sal:25000, status: “N" })

4.Update a document into collection

db. Teacher_info.update({ sal: { $gt: 25000 } }, { $set: { Dept_name: “ETC" } }, { multi: true })

db. Teacher_info.update({ status: "A" } , { $inc: { sal: 10000 } }, { multi: true })

5.Remove a document from collection

db.Teacher_info.remove({Teacher_id: "pic001"});
db. Teacher_info.remove({})

6.Alter a field into a mongodb document

db.Teacher_info.update({ }, { $set: { join_date: new Date() } }, { multi: true})

7.To drop a particular collection

db.Teacher_info.drop()

REFERENCE BOOK:

Kristina Chodorow, MongoDB The definitive guide, O'Reilly Publications, ISBN:978-93-5110-269-4,2nd Edition.

CONCLUSION:

Understand to implement data from mongodbdatabase with the help of statement and operators.

Assignment: 13

AIM: Execute at least 10 queries on above MongoDB database that demonstrates following querying techniques:

- Find
- FindOne (specific values)
- Conditional queries
  (Query conditionals, OR queries, $not, Conditional semantics)
- Type-specific queries

(Null, Regular expression, Querying arrays)

PROBLEM STATEMENT /DEFINITION

Execute at least 10 queries on above MongoDB database that demonstrates following querying techniques:

- Find
- FindOne (specific values)
- Conditional queries
  (Query conditionals, OR queries, $not, Conditional semantics)
- Type-specific queries

  (Null, Regular expression, Querying arrays)

OBJECTIVE:

To understand Mongodbretrievalcommands
To implement the concept of document oriented databases.

THEORY:

When we retrieve a document from mongodb collection it always add a _id field in the every document which conatin unique _id field.

ObjectId(<hexadecimal>)

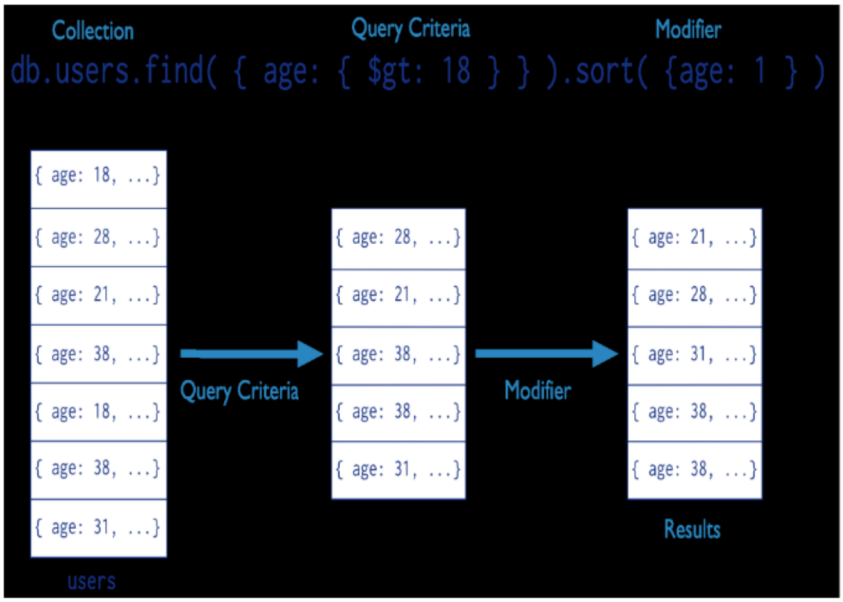
Returns a new ObjectId value. The 12-byte ObjectId value consists of:

4-byte value representing the seconds since the Unix epoch,

3-byte machine identifier,

2-byte process id, and

3-byte counter,starting with a random value.



1. Retrieve a collection in mongodb using Find command

db.Teacher.find()

```
{ "_id" : 101, "Name" : "Dev",
"Address" : [  {  "City" : "Pune",  "Pin" : 444043 } ],
"Department" : [  {  "Dept_id" : 111,  "Dept_name" : "IT" } ],
 "Salary" : 78000 }

{ "_id" : 135, "Name" : "Jennifer", "Address" : [  {  "City" :
"Mumbai",  "Pin" : 444111 } ], "Department" : [  {  "Dept_id" : 112,
"Dept_name" : "COMP" } ], "Salary" : 65000 }
{ "_id" : 126, "Name" : "Gaurav", "Address" : [  {  "City" : "Nashik",
"Pin" : 444198 } ], "Department" : [  {  "Dept_id" : 112,  "Dept_name"
: "COMP" } ], "Salary" : 90000 }
{ "_id" : 175, "Name" : "Shree", "Address" : [  {  "City" : "Nagpur",
"Pin" : 444158 } ], "Department" : [  {  "Dept_id" : 113,  "Dept_name"
: "ENTC" } ], "Salary" : 42000 }
{ "_id" : 587, "Name" : "Raman", "Address" : [  {  "City" : "Banglore",
"Pin" : 445754 } ], "Department" : [  {  "Dept_id" : 113,  "Dept_name"
: "ENTC" } ], "Salary" : 79000 }
{ "_id" : 674, "Name" : "Mandar", "Address" : [  {  "City" : "Jalgaon",
"Pin" : 465487 } ], "Department" : [  {  "Dept_id" : 111,  "Dept_name"
: "IT" } ], "Salary" : 88000 }
{ "_id" : 573, "Name" : "Manish", "Address" : [  {  "City" : "Washim",
"Pin" : 547353 } ], "Department" : [  {  "Dept_id" : 112,  "Dept_name"
: "COMP" } ], "Salary" : 65000 }
```

2. Retrieve a document from collection in mongodb using Find command using condition

>db.Teacher_info.find({sal: 25000})

3. Retrieve a document from collection in mongodb using Find command using or operator

>db.Teacher_info.find({ $or: [{ status: "A" } , { sal:50000 }] })

4. Retrieve a document from collection in mongodb using Find command using greater than , less than, greater than and equal to ,less than and equal to operator

>db. Teacher_info.find({ sal: **{ $gt:** 40000 } })

>db.media.find({ Released : **{$gt :** 2000} }, { **"Cast" : 0** })

{ "_id" : ObjectId("4c4369a3c603000000007ed3"), "Type" : "DVD", "Title" :"Toy Story 3", "Released" : 2010 }

>db.media.find ({ Released : **{$gte** : 1999 } }, { "Cast" : 0 })

{ "_id" : ObjectId("4c43694bc603000000007ed1"), "Type" : "DVD", "Title" :"Matrix, The", "Released" : 1999 }

{ "_id" : ObjectId("4c4369a3c603000000007ed3"), "Type" : "DVD", "Title" :"Toy Story 3", "Released" : 2010 }

>db.media.find ({ Released : **{$lt :** 1999 } }, { "Cast" : 0 })

{ "_id" : ObjectId("4c436969c603000000007ed2"), "Type" : "DVD", "Title" : "Blade Runner", "Released" : 1982 }

>db.media.find({Released : **{$lte:** 1999}}, { "Cast" : 0 })

{ "_id" : ObjectId("4c43694bc603000000007ed1"), "Type" : "DVD", "Title" :"Matrix, The", "Released" : 1999 }

{ "_id" : ObjectId("4c436969c603000000007ed2"), "Type" : "DVD", "Title" :"Blade Runner", "Released" : 1982 }

>db.media.find({Released : **{$gte: 1990, $lt : 2010**}}, { "Cast" : 0 })

{ "_id" : ObjectId("4c43694bc603000000007ed1"), "Type" : "DVD", "Title" :"Matrix, The", "Released" : 1999 }

Retrieval a value from document which contain array field

Exact Match on an Array

db.inventory.find({ tags: ['fruit', 'food', 'citrus'] })

Match an Array Element

db.inventory.find({ tags: 'fruit' })

Match a Specific Element of an Array

db.inventory.find({ 'tags.**0**' : 'fruit' })

6.MongoDB provides a db.collection.findOne() method as a special case of find() that returns a single document.

7.Exclude One Field from a Result Set

>db.records.Find({ "user_id": { $lt: 42} }, { history: 0})

8.Return Two fields and the _id Field

>db.records.find({ "user_id": { $lt: 42} }, { "name": 1, "email": 1})

9.Return Two Fields and Exclude _id

>db.records.find({ "user_id": { $lt: 42} }, { "_id": 0, "name": 1 , "email": 1 })

10. Retrieve a collection in mongodb using Find command and pretty appearance

>db.<collection>.find().pretty()

REFERENCE BOOK:

Kristina Chodorow, MongoDB The definitive guide, O'Reilly Publications, ISBN:978-93-5110-269-4,2nd Edition.

CONCLUSION:

Understand to implement data from mongodbdatabase with the help of statement and operators.

Assignment: 14

AIM:
Execute at least 10 queries on any suitable MongoDB database that demonstrates following:

- $ where queries
- Cursors (Limits, skips, sorts, advanced query options)
- Database commands

PROBLEM STATEMENT /DEFINITION

Execute at least 10 queries on any suitable MongoDB database that demonstrates following:

- $ where queries
- Cursors (Limits, skips, sorts, advanced query options)
- Database commands

OBJECTIVE:

To understand Mongodb retrieval commands
To implement the concept of document oriented databases.

THEORY:

```
db.users.find(                    collection
    {age:{$gt:18}},               query criteria
    {name :1,address:1}           projection
}.limit(5)                        cursor modifier
```

Retrieve a document in ascending or descending order using 1 for ascending and -1 for descendingfrom collection in mongodb

```
>db. Teacher_info.find( { status: "A" } ).sort( {sal: -1 } )

>db.audit.find().sort( { $natural: -1 } ).limit ( 10 )

>db.Employee.find().sort({_id:-1})
```

```
{ "_id" : 106, "Name" : "RAJ", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" : "ACCOUNTING"
} ], "Salary" : 50000 }
{ "_id" : 105, "Name" : "ASHOK", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" : "ACCOUNTING"
} ], "Salary" : 40000 }
{ "_id" : 104, "Name" : "JOY", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ],
"Salary" : 20000 }
```

```
{ "_id" : 103, "Name" : "RAM", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ],
"Salary" : 10000 }
{ "_id" : 102, "Name" : "AKASH", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 80000 }
{ "_id" : 101, "Name" : "Dev", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 78000 }
```

>db.Employee.find().sort({_id:1})

```
{ "_id" : 101, "Name" : "Dev", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 78000 }
{ "_id" : 102, "Name" : "AKASH", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 80000 }
{ "_id" : 103, "Name" : "RAM", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ],
"Salary" : 10000 }
{ "_id" : 104, "Name" : "JOY", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ],
"Salary" : 20000 }
{ "_id" : 105, "Name" : "ASHOK", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" : "ACCOUNTING"
} ], "Salary" : 40000 }
{ "_id" : 106, "Name" : "RAJ", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" : "ACCOUNTING"
} ], "Salary" : 50000 }
>db.Employee.find().sort({$natural:-1}).limit(2)
{ "_id" : 106, "Name" : "RAJ", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" : "ACCOUNTING"
} ], "Salary" : 50000 }
{ "_id" : 105, "Name" : "ASHOK", "Address" : [ { "City" : "NASIK", "Pin" :
411002 } ], "Department" : [ { "Dept_id" : 113, "Dept_name" : "ACCOUNTING"
} ], "Salary" : 40000 }
```

>db.Employee.find().sort({$natural:1}).limit(2)

```
{ "_id" : 101, "Name" : "Dev", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 78000 }
{ "_id" : 102, "Name" : "AKASH", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 111, "Dept_name" : "HR" } ],
"Salary" : 80000 }
>db.Employee.find({Salary:{$in:[10000,30000]}})
{ "_id" : 103, "Name" : "RAM", "Address" : [ { "City" : "Pune", "Pin" :
444043 } ], "Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ],
"Salary" : 10000 }
>db.Employee.update({"Name":"RAM"},{ $set :{Address:{City: "Nasik"}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>db.Employee.find({"Name":"RAM"})
```

```
{ "_id" : 103, "Name" : "RAM", "Address" : { "City" : "Nasik" },
"Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ], "Salary" :
10000 }
>db.Employee.update({"Name":"RAM"},{$inc :{"Salary": 10000 } })
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>db.Employee.find({"Name":"RAM"})
{ "_id" : 103, "Name" : "RAM", "Address" : { "City" : "Nasik" },
"Department" : [ { "Dept_id" : 112, "Dept_name" : "SALES" } ], "Salary" :
20000 }
```

Retrieve documentwith a particular from collection in mongodb

>db.Employee.find().limit(2).pretty()

```
{
	"_id" : 101,
	"Name" : "Dev",
	"Address" : [
		{
			"City" : "Pune",
			"Pin" : 444043
		}
	],
	"Department" : [
		{
			"Dept_id" : 111,
			"Dept_name" : "HR"
		}
	],
	"Salary" : 78000
}
{
	"_id" : 102,
	"Name" : "AKASH",
	"Address" : [
		{
			"City" : "Pune",
			"Pin" : 444043
		}
	],
	"Department" : [
		{
			"Dept_id" : 111,
			"Dept_name" : "HR"
		}
	],
	"Salary" : 80000
}
```

Retrieve document skipping some documents from collection in mongodb

>db.Employee.find().skip(3).pretty()

```
{
	"_id" : 104,
	"Name" : "JOY",
	"Address" : [
		{
			"City" : "Pune",
			"Pin" : 444043
		}
	],
	"Department" : [
		{
			"Dept_id" : 112,
			"Dept_name" : "SALES"
		}
	],
	"Salary" : 20000
}
{
	"_id" : 105,
	"Name" : "ASHOK",
	"Address" : [
		{
			"City" : "NASIK",
			"Pin" : 411002
		}
	],
	"Department" : [
		{
			"Dept_id" : 113,
			"Dept_name" : "ACCOUNTING"
		}
	],
	"Salary" : 40000
}
{
	"_id" : 106,
	"Name" : "RAJ",
	"Address" : [
		{
			"City" : "NASIK",
			"Pin" : 411002
		}
	],
	"Department" : [
		{
			"Dept_id" : 113,
			"Dept_name" : "ACCOUNTING"
		}
	],
	"Salary" : 50000
}
```

REFERENCE BOOK:

Kristina Chodorow, MongoDB The definitive guide, O'Reilly Publications, ISBN:978-93-5110-269-4,2nd Edition.

CONCLUSION:

Understand to implement data from mongodbdatabase with the help of statement and operators

Assignment: 15

AIM: Implement Map reduces operation with suitable example on above MongoDB database

PROBLEM STATEMENT /DEFINITION

Implement Map reduces operation with suitable example on above MongoDB database

OBJECTIVE:

To understand the concept of Mapreduce in mongodb.
To implement the concept of document oriented databases.

THEORY:

- Implements the MapReduce model for processing large data sets.
- Can choose from one of several output options (inline, new collection, merge, replace, reduce)
- MapReduce functions are written in JavaScript.
- Supports non-sharded and sharded input collections.
- Can be used for incremental aggregation over large collections.
- MongoDB 2.2 implements much better support for sharded map reduce output.
- Map/Reduce involves two steps:
- first, map the data from the collection specified;
- second, reduce the results.

>db.createCollection("Order")

- { "ok" : 1 }

>db.order.insert({cust_id:"A123",amount:500,status:"A"})

- WriteResult({ "nInserted" : 1 })

>db.order.insert({cust_id:"A123",amount:250,status:"A"})

- WriteResult({ "nInserted" : 1 })

>db.order.insert({cust_id:"B212",amount:200,status:"A"})

- WriteResult({ "nInserted" : 1 })

>db.order.insert({cust_id:"A123",amount:300,status:"d"})

- WriteResult({ "nInserted" : 1 })

- **Map Function**

- var mapFunction1 = function()

- { emit(this.cust_id, this.amount);};

- **Reduce Function**

- var reduceFunction1 = function(key, values)
- {return Array.sum(values); };

db.order.mapReduce

(mapFunction1, reduceFunction1, {query: {status: "A" },

out: "order_totals"});

```
"result" : "order_totals",
"timeMillis" : 28,
"counts" : {
        "input" :3,
        "emit" :3,
        "reduce" : 1,
        "output" : 2
},
"ok" : 1,}
```

>db.order.mapReduce(

Map Function -> function() { emit(this.cust_id, this.amount);},

Reduce Function -> function(key, values) { return Array.sum (values)},

Query à {query: { status:"A"},

Output collection à out: "order_ totals"})

```
{
        "result" : "order_totals",
        "timeMillis" : 27,
        "counts" : {
```

```
        "input" : 3,
        "emit" : 3,
        "reduce" : 1,
        "output" : 2
    },
    "ok" : 1,
}
```

To display result of mapReduce function use collection created in OUT.

Db.<collection name>.find();

db.order_totals.find();

{ "_id" : "A123", "value" : 750 }

{ "_id" : "B212", "value" : 200 }

REFERENCE BOOK:

Kristina Chodorow, MongoDB The definitive guide, O'Reilly Publications, ISBN:978-93-5110-269-4,2nd Edition.

CONCLUSION:

Understand to mapreduce operation in mongodb

Assignment: 16

AIM: Implement the aggregation and indexing with suitable example on above MongoDB database.Demonstrate Following

- Aggregation framework
- Create and drop different types of indexes and explain () to show the advantage of the indexes.

PROBLEM STATEMENT /DEFINITION

Implement the aggregation and indexing with suitable example on above MongoDB database. Demonstrate Following

- Aggregation framework
- Create and drop different types of indexes and explain () to show the advantage of the indexes.

OBJECTIVE:

- To understand the concept of Aggregation in mongodb.
- To implement the concept of document oriented databases.

THEORY:

- New feature in the Mongodb2.2.0 production release (August, 2012).
- Designed with specific goals of **improving performance** and **usability.**
- Returns result set inline.
- Supports **non-sharded and sharded**input collections.
- Uses a **"pipeline"** approach where objects are transformed as they pass through a series of pipeline operators such as matching, projecting, sorting, and grouping.
- **Pipeline operators need not produce one output document for every input document:** operators may also generate new documents or filter out documents.

Implementation of Aggregation:-

> use Teacher
switched to db Teacher
>db.Teacher.find()

{ "_id" : 101, "Name" : "Dev", "Address" : [{ "City" : "Pune", "Pin" : 444043 }],
"Department" : [{ "Dept_id" : 111, "Dept_name" : "IT" }], "Salary" : 78000 }
{ "_id" : 135, "Name" : "Jennifer", "Address" : [{ "City" : "Mumbai", "Pin" : 444111 }],
"Department" : [{ "Dept_id" : 112, "Dept_name" : "COMP" }], "Salary" : 65000 }
{ "_id" : 126, "Name" : "Gaurav", "Address" : [{ "City" : "Nashik", "Pin" : 444198 }],
"Department" : [{ "Dept_id" : 112, "Dept_name" : "COMP" }], "Salary" : 90000 }
{ "_id" : 175, "Name" : "Shree", "Address" : [{ "City" : "Nagpur", "Pin" : 444158 }],
"Department" : [{ "Dept_id" : 113, "Dept_name" : "ENTC" }], "Salary" : 42000 }
{ "_id" : 587, "Name" : "Raman", "Address" : [{ "City" : "Banglore", "Pin" : 445754 }],
"Department" : [{ "Dept_id" : 113, "Dept_name" : "ENTC" }], "Salary" : 79000 }
{ "_id" : 674, "Name" : "Mandar", "Address" : [{ "City" : "Jalgaon", "Pin" : 465487 }],
"Department" : [{ "Dept_id" : 111, "Dept_name" : "IT" }], "Salary" : 88000 }
{ "_id" : 573, "Name" : "Manish", "Address" : [{ "City" : "Washim", "Pin" : 547353 }],
"Department" : [{ "Dept_id" : 112, "Dept_name" : "COMP" }], "Salary" : 65000 }

>db.Teacher.aggregate([
... {$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}}
...])

```
{
        "result" : [
                {
                        "_id" : [
                                {
                                        "Dept_id" : 113,
                                        "Dept_name" : "ENTC"
                                }
                        ],
                        "totalsalary" : 121000
                },
                {
                        "_id" : [
                                {
                                        "Dept_id" : 112,
                                        "Dept_name" : "COMP"
                                }
                        ],
                        "totalsalary" : 220000
                },
                {
                        "_id" : [
                                {
                                        "Dept_id" : 111,
                                        "Dept_name" : "IT"
                                }
                        ],
                        "totalsalary" : 166000
```

```
                }
        ],
        "ok" : 1
}
>db.Teacher.aggregate([
{$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}},{$group:{_id:"$_id.Departm
ent",AvgSal:{$sum:"$totalsalary"}}}])
{ "result" : [ { "_id" : [ ], "AvgSal" : 507000 } ], "ok" : 1 }
>db.Teacher.aggregate([
{$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}},{$match:{totalsalary:{$gte:2
00000}}}])
{
        "result" : [
                {
                        "_id" : [
                                {
                                        "Dept_id" : 112,
                                        "Dept_name" : "COMP"
                                }
                        ],
                        "totalsalary" : 220000
                }
        ],
        "ok" : 1
}
>db.Teacher.aggregate([ {$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}}, {
$sort:{totalsalary:1}}])
{
        "result" : [
                {
                        "_id" : [
                                {
                                        "Dept_id" : 113,
                                        "Dept_name" : "ENTC"
                                }
                        ],
                        "totalsalary" : 121000
                },
                {
                        "_id" : [
                                {
                                        "Dept_id" : 111,
                                        "Dept_name" : "IT"
                                }
                        ],
                        "totalsalary" : 166000
```

```
                },
                {
                        "_id" : [
                                {
                                        "Dept_id" : 112,
                                        "Dept_name" : "COMP"
                                }
                        ],
                        "totalsalary" : 220000
                }
        ],
        "ok" : 1
}
```

>db.Teacher.aggregate([{$group:{_id:"$Department",totalsalary:{$sum:"$Salary"}}}, { $group: { _id:"$_id.Department", big: { $last: "$_id.Dept_name" }, bigsalary: { $last:"$totalsalary"}, small: { $first:"$_id.Dept_name"}, smallsalary: { $first:"$totalsalary"} }}])

```
{
        "result" : [
                {
                        "_id" : [ ],
                        "big" : [
                                "IT"
                        ],
                        "bigsalary" : 166000,
                        "small" : [
                                "ENTC"
                        ],
                        "smallsalary" : 121000
                }
        ],
        "ok" : 1
}
```

REFERENCE BOOK:

Kristina Chodorow, MongoDB The definitive guide, O'Reilly Publications, ISBN:978-93-5110-269-4,2nd Edition.

CONCLUSION:

Understand to aggregation operation in mongodb

GROUP D

MINI PROJECT

OR

DATABASE APPLICATION DEVELOPMENT

Assignment: 17

AIM: Design and Implement Database Mini Project**.**

PROBLEM STATEMENT /DEFINITION

Design and Implement any Database Application using Java/PHP/Python etc. and MySQL/MongoDB (preferably MySQL) as a back end. Implement Database navigation operations (add, delete, edit etc.) using ODBC/JDBC. Use stored procedure, Trigger and functions.

OBJECTIVE:

1. To understand applications of DBMS by implementing mini project.
2. To learn effective UI designs.
3. To learn to design & implement database system for specific domain.
4. To learn to design system architectural & flow diagram.
5. To learn to draw ER diagram.
6. To learn to convert ER diagram to DB tables.
7. To learn to generate various report useful for analysis.

Mini Project Report Format:

Abstract

Acknowledgement

List of Tables & Figures

Contents

1. **Introduction**
   - **1.1 Purpose**
   - **1.2 Scope**
   - **1.3 Definition, Acronym, and Abbreviations**
   - **1.4 References**
   - **1.5 Developers' Responsibilities: An Overview**
2. **General Description**
   - **2.1 Product Function Perspective**
   - **2.2 User Characteristics.**
   - **2.4 General Constraints**

2.5 Assumptions and Dependencies

3. Specific Requirements

3.1 Inputs and Outputs

3.2 Functional Requirements

3.3 Functional Interface Requirements

3.3 Performance Constraints

3.4 Design Constraints

3.6 Acceptance criteria

4. System Design

4.1 ER Model

4.2 Schema Description

4.3 Tables Description

4.4 System Flow chart / Activity diagram

4.5 User Interface Design

4.6 Error Messages / Alerts Design

4.7 Test Case Design

5. System Implementation

5.1 Hardware and Software Platform description

5.2 Tools used

5.3 System Verification and Testing (Test Case Execution)

5.4 Future work / Extension

5.5 Conclusion

References

Annexure:

A. GUIs / Screen Snapshot of the System Developed
B. Part –A: SQL and PL/SQL, Triggers, Stored Procedures, Functions