REVA UNIVERSITY

BENGALURU, INDIA

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

# MODERN DATABASES LAB
# BT20EF0508

# for

## Fifth Semester
## B.Tech in Computer Science and Engineering

**(Prepared in AUG-2022)**

| | |
|---|---|
| **Name** | |
| **SRN** | |
| **Branch** | |
| **Semester** | |
| **Section** | |

| Academic Year | |
|---|---|

# INDEX

## List of Experiments

| No | Title of the Experiment | Tools and Techniques | Expected Skill /Ability |
|---|---|---|---|
| | **Part-A** | | |
| 1. | **a) Product - Order System**<br><br>In recent years, most of the grocery items are available online; hence people are doing online transactions for purchase. There are lot of discounts and benefits through the online orders. Since everyone in the life is busy with one or other works, such applications will save their time.<br><br>These online transaction based applications require many databases to be built for storage and transaction management. Design a product-order database which can store the details of customers, agents and the products. All the details of sold products along with commission from different agents across different cities<br><br>will get stored in this database and utilized for<br><br>transactions.<br><br>**To create DDL commands**<br><br>Customer (cid, cname, city, discount)<br><br>Agent (aid, aname, city, commission)<br><br>Product (pid, pname, city, quantity, price)<br><br>Orders (ordno, month, cid, aid, pid, qty, amount) | **SQL / Oracle** | Create and perform operations. |

| | | | |
|---|---|---|---|
| | **b)To practice with the DML Commands** | **SQL / Oracle** | Create and perform operations on Tuples |
| 2 | **a) Queries**<br><br>a. Retrieve the customer ids of any product which has been ordered by agent "a06".<br><br>b. Retrieve cities in which customers or agents located.<br><br>c. List product ids which have been ordered by agents from the cities “Dargeling” or “Srinagar”.<br><br>d. Retrieve customer ids whose discounts are less than the maximum discount.<br><br>e. Retrieve product ids ordered by at least two<br><br>customers.<br><br>f. For each (aid, pid) pair get the sum of the orders aid has placed for pid.<br><br>g. Retrieve product ids and total quantity ordered for each product when the total exceeds 1000.<br><br>h. List the names of the customers and agent who<br><br>placed an order through that agent.<br><br>i. Retrieve order numbers placed by customers in<br><br>"Dargeling" through agents in "New Delhi".<br><br>j. Retrieve names of the customers who have the same discount as that of any (one) of the customers in "Dargeling" or "Bangalore". | **SQL / Oracle** | Create and perform operations. |

| | | | |
|---|---|---|---|
| | **b) Queries**<br><br>a. Retrieve customer ids with smaller discounts than every customer from " Srinagar"<br>b. Retrieve names of the customers who have placed an order through agent "a05". (usingexists) m. Retrieve names of the customers who do not place orders through agent "a05". (using not exists) n. Retrieve customer ids whose orders placed through all the agents in "New Delhi".<br>c. Retrieve agent ids either from "New Delhi" or<br>d. "Srinagar" who place orders for ALL products priced over one dollar.<br>e. Retrieve names and ids of the customers and agents along with total dollar sales for that pair. Order the result from largest to smallest total sales. Also retain only those pairs for which total dollar sales is at least 9000.00.<br>f. Increase the percent commission by 50% for all agents in "New York".<br>g. Retrieve the total quantity that has been placed for each product. | **SQL / Oracle** | Create and perform operations |
| 3. | **a) Employee Database System**<br><br>Design a company database which can store the details of Departments, projects, their Employee and his / her dependent details of a particular organization.<br><br>***To create DDL command*** for the following:<br><br>**Employee** (ssn, name, salary, sex, super_ssn, address,dno)<br><br>**Department** (dname, dnumber,mgr_ssn)<br><br>**Dept_Loc** ( dnumber, dloc)<br><br>**Project** (pname, pnumber, plocation, dnum) | **SQL / SQL Server** | Create and perform operations |

| | | | |
|---|---|---|---|
| | **Works_On**(essn, pno, hours)<br><br>**Dependent** (essn, depen_name, address, relationship,sex) | | |
| | **b) Queries**<br><br>a. **Retrieve the names of the Employees** who works on all the projects controlled by dept no 3.<br><br>b. Retrieve the names of the Employees who gets<br><br>second highest salary.<br><br>c. Retrieve the names of the Employees who have no dependents in alphabetical order.<br><br>d. List the names of all Employees with at least two dependents.<br><br>e. Retrieve the number of Employees and their average salary working in each Department.<br><br>f. Retrieve the highest salary paid in each Department in descending order.<br><br>g. Retrieve the SSN of all Employees who work on at least one of the project numbers 1, 2, 3.<br><br>h. Retrieve the number of dependents for an Employee named RAM.<br><br>i. Retrieve the names of the managers working in<br><br>location named xyz who has no female dependents.<br><br>j. Retrieve the names of the Employees who works in the same Department as that of RAM. | **SQL / SQL Server** | |

| | | | |
|---|---|---|---|
| 4 | **Create an employee database with the fields: {eid, ename, dept, desig, salary, yoj, address {dno, street, locality, city}}** | **MongoDB** | Create and perform operations |
| 5. | **Create a Book Data Base with the fields: (ISBN, bname, author [], year, publisher, price) use Book.** | **MongoDB** | Create and perform operations |
| 6. | **Create a Food Database with the fields: (food id, food cat, food name, chef name[ ], price, ingredients [], hotel name, hotel address {no, street, locality, city})** | **MongoDB** | Create and perform operations |
| **Part-B - Mini Project *E-Commerce Management System*** | | | |
| | In this modern era of online shopping no seller wants to be left behind, moreover due to its simplicity the shift from offline selling model to an online selling model is witnessing a rampant growth. Therefore, as an engineer our job is to ease the path of this transition for the seller. Amongst many things that an online site requires the most important is a database system. Hence in this project we are planning to design a database where small clothing sellers can sell their product online.<br><br>**Modules:**<br><br>User<br>Add user<br>Delete user<br>Search user | **MySQL:**<br><br>SQL commands<br>PL/SQL Triggers | Draw the E-R diagram and to perform its operations.<br><br>**Implementation:**<br>o Creating tables<br>o Inserting data<br><br>**Queries**<br><br>Basic queries<br><br>PL/SQL<br><br>Trigger |

| | | | |
|---|---|---|---|
| | Modify user<br>Transaction<br><br><br><br>Place order<br>Deliver the order<br>View the transactions<br>Product<br><br><br>Add product<br>Delete product<br>Search product<br>View product<br><br><br>• A new user can register on the website.<br>• A customer can see details of the product present in the cart<br>• A customer can view his order history.<br>• Admin can start a sale with certain discount on every product.<br>• Customer can filter the product based on the product details.<br>• A customer can add or delete a product from the cart.<br>• A seller can unregister/ stop selling his product. | | function |

| | | | |
|---|---|---|---|
| | • A seller/ customer can update his details.<br>• Admin can view the products purchased on particular date.<br>• Admin can view number of products sold on a particular date.<br>• A customer can view the total price of product present in the cart unpurchased.<br>• Admin can view details of customer who have not purchased anything.<br>• Admin can view total profit earned from the website. | | |
| | | | |

## 1. Lab Objectives:

The objectives of this course are to:

1. Explain database applications, data models, schemas and instances.
2. Demonstrate the use of constraints and relational algebra operations.
3. Emphasize the importance of normalization in databases.
4. Familiarize issues of concurrency control and transaction management

## 2. Lab Outcomes:

On successful completion of this course; student shall be able to:

- Analyze the requirements of a given database problem and design viable ERModels.
- Create database schemas, select and apply suitable integrity constraints for querying databases using SQL interface.
- Apply DDL, DML commands on different database schema.
- Develop database applications such as Product Order System, Employee Database System using modern tools(mongoDB).
- Develop and interpret PL/SQL blocks to centralize database applications for maintainability and reusability.
- Design and develop database application based on real life problem to address societal issues using various database management tools.

**3. Lab Requirements:**

Following are the required hardware and software for this lab, which is available in the laboratory.

- **Hardware:** Desktop system or Virtual machine in a cloud with OS installed. Presently in the Lab, Pentium IV Processor having 1 GB RAM and 250 GB Hard Disk is available. Desktop systems are dual boot having Windows as well as Linux OS installed on them.
- **Software:** The DBMS packages that fall in this category are as follows:
    - Oracle ( follows 7 rules )
    - DB2 ( follows 9 rules )
    - Ingress ( follows 10 rules )
    - Sybase ( follows 9 rules )

**Log Into Oracle**

**Microsoft Windows**

Under Windows environment, the Oracle client is called SQL*Plus.The following are Steps for logging into the SQL.

Steps: 1. Click **Start**, and then click **Run**.

2. Type **sqlplus**, and fill in the username, password, and database name
3. After you log in to SQL*Plus, you see the following message:

Connected to: Oracle10g Enterprise Edition Release 9.1.7.0.0 - Production

JServer Release 9.1.7.0.0 – Production and you should receive the prompt:

**Creating user:** Create user <yourName> identified by <Password>;

Where <yourName> is again your login name, and <Password> is the password you would like to use in the future. This command, like all other SQL commands, should be terminated with a semicolon.

**Changing Your Password** : In response to the SQL> prompt, type

Alter user <username> identified by <Password>;

where <username> is again your login name, and <Password> is the password you would like to use in the future. This command, like all other SQL commands, should be terminated with a semicolon.

## 4. Guidelines to Students

- Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
- Students are required to carry their observation / programs book with completed exercises while entering the lab.
- Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- Lab records need to be submitted on or before date of submission.
- Students are not supposed to use flash drives.
- In C3 exam one Data base application will be asked and the set of some queries will be given in the final exam and evaluated for 50 marks and scale down to 25 marks.

## 5. Introduction to Database, Oracle and SQL

**Database**

A group of tables with related data in them are called database.

**Database Management System**

DBMS consists of a collection of interrelated data and a set of programs to manage these data.

**Data Model**

Structure of database is defined by data model.

Different data models are as follows:

- Object Oriented model
- Relational model
- Network model
- Hierarchical model

**Relational model**

- Relational model uses a collection of tables to represent both data and relationship among those tables.
- Most database management systems are based on the relational model.
- RDBMS follows codd's rules.
- There are 12 rules specified by E.F. Codd that must be satisfied by adatabase package for being an RDBMS.

## SQL * Plus

**SQL *Plus** enables you to manipulate SQL commands and PL/SQL blocks, and to perform many additional tasks as well. Through SQL *Plus, you can

- Enter, edit, store, retrieve, and SQL commands and PL/SQL blocks.
- Format, perform calculations on, store, and print query results in the form of reports.
- List column definitions for any table.
- Access and copy data between SQL databases.
- Send messages to and accept responses from an end user.

## Oracle Overview

Oracle is one the most popular *Relational Database Management System* (RDBMS). Some other famous RDBMS includes Microsoft SQL Server, Sybase, MySQL, PostgreSQL, etc. Essentially, all the aforementioned RDBMS employs *Structural Query Language* (SQL) as their query interface. Users usually issue their queries by SQL through a "client". Different RDBMS offer different forms of clients. For example, MS SQL Server offers a GUI interface for user to type in their SQL language, and their queries would be executed after pressing the "Execute" button on the client. Oracle provides both GUI client and command-line client. In this lesson, we will study the command-line client, *SQL*Plus*. In addition, Oracle extends the standard SQL (e.g. select * from table) with its application-specific commands (e.g. checking how many table you have been created in your Oracle account) into a Oracle specific language called *PL/SQL*. In this tutorial, you will interact with Oracle database, thru SQL*Plus, by issuing a number of PL/SQL queries.

## SQL Basics

**Structured Query Language (SQL)**, which is an ANSI standard language for interacting with relational databases, is the main tool for extracting the information.

A **database** is a representation of a real-world thing called an **Entity.** Examples of entities are vehicles, Employees, customers, fish, buildings, and even things such as baseball teams. The database stores facts about the entity in an organized framework, model, or schema. These facts are called **attributes.**

An **Instance** is one occurrence of an entity.

Each entity must have an identifier, which is one or more attributes that make each entity instance unique from any other instance. The identifier should contain a value that does not change.

Examples of identifiers are student IDs, payroll numbers, or social security numbers.

**Primary key** - If the entity does not have an attribute that can be used as an identifier, an artificial identifier can be created. The identifier on an entity is often called a **primary key.**

**Foreign key -** A foreign key is a set of attributes of the considered table that exists as a primary key attributes in another table. Database records are matched (joined) through the use of primary and foreign keys.

**Normalization -** Normalization **is** a process consisting of series of steps, which is used to group the database attributes. The purpose of this design is to ensure that the tables within the database are space efficient and performance efficient.

- **Zero Normal Form** - Each of the relations (tables) has a unique identifier (primary key).
- **First Normal Form** - Separate the repeating groups of attributes or multi valued attributes into a relation of their own. Be sure to form composite keys.
- **Second Normal Form** - Establish full functional dependency by separating out attributes that are not fully dependent on the full primary keys.
- **Third Normal Form** - Remove transitive dependencies by separating attributes that are dependent on a non key attribute.

**How SQL works**

The purpose of SQL is to interface to a relational database such as Oracle, and all SQL statements are **instructions to the databases.**

SQL provides commands for a variety of tasks including:

- Querying data
- Inserting, updating, and deleting rows in a table
- Creating, replacing, altering, and dropping objects
- Controlling access to the database and its objects
- Guaranteeing database consistency and integrity

## Data Types

Each literal or column value manipulated by Oracle has a data type. A value's data type associates a fixed set of properties with the value. These properties cause Oracle to treat values of one data types differently values of another.

**Character Data types** - Character data types are used to manipulate words and freeform text. These data types are used to store character. These data types are used for character data:

- **CHAR** - The **CHAR** data type specifies a fixed length character is 1 character and maximum allowed is 2000 character.
- **NCHAR** - The **NCHAR** data types specifies a fixed-length national character set character string. The maximum column size allowed is 2000 bytes.

- **NVARCHAR2** - The **NVARCHAR2** data type specifies variable-length national character string. The maximum column size allowed is 4000 bytes.
- **VARCHAR2** - The **VARCHAR2** data type specifies a variable length character string. The maximum length of VARCHAR2 data is 4000 bytes.

**Number Data type -** The **NUMBER** data type is used to store zero, positive and negative fixed and floating point numbers with magnitudes

**Floating Point Numbers -** A floating point value either can have a decimal point anywhere from the first to the last digit or can omit the decimal point altogether.

**Long Data type -** LONG columns store variable length variable length character strings containing up to 2 gigabytes, or $2^{31}$ -1 bytes. LONG data type is subject to some restrictions which are:

- A table cannot contain more than one LONG column.
- LONG columns cannot appear in integrity constraints.
- LONG columns cannot be indexed.

Also, LONG columns cannot appear in certain parts of SQL statements:

- WHERE, GROUP BY, or CONNECT BY clause or with the DISTINCT operator in SELECT statements.
- UNIQUE clause of a SELECT statement.
- Select list of queries containing GROUP BY clauses.
- Select list of sub queries or queries combined by set operators.

**DATE Data type -** The DATE data types is used to store date and time information.

**Operators** - All the normal Arithmetic, Relational, Logical operators are used in SQL.

**SQL Commands -** In order to define schemas, store data, retrieve data and to make amendments in schema and data stored in the database different types of commands are used which are:

- Data Definition Language Commands (DDL)
- Data Manipulation Language Commands (DML)
- Transaction Control Commands (TCL)
- Session Control Commands (SCL)
- System Control Commands (SCC)

**Data Definition Language (DDL)** commands allow you to perform these tasks:

- Create, Alter, and Drop schema objects(CAD)
- Grant and Revoke privileges and roles
- Analyses information on a table, index, or cluster
- Establish auditing options
- Add comments to the data dictionary

## Create Table Command

It defines each column of the table uniquely.
Each column has minimum of three attributes, a name , data type and size.

**Syntax:** **Create table** <table name> (<col1> <datatype>(<size>),<col2> <datatype><size>)) ;

**Ex:** Create table emp (empno number(4) primary key, ename char(10)) ;

## Modifying the structure of tables

**a) Add new columns**

**Syntax :** **Alter table** <tablename> **add**(<new col><datatype(size),<new col>datatype(size));

**Ex:** **Alter table** emp **add**(sal number(7,2)) ;

**b) Dropping a column from a table**

**Syntax:** **Alter table** <tablename> **drop** column <col> ;

**Ex:** **Alter table** emp **drop** column sal ;

**c) Modifying existing columns**

**Syntax: Alter table** <tablename> **modify** (<col><newdatatype>(<newsize>)) ;

**Ex:** Alter table emp modify(ename varchar2(15)) ;

**d) Renaming the tables**

**Syntax: Rename** <oldtable> **to** <new table> ;

**Ex:** rename emp to emp1 ;

**Truncating the tables**

**Syntax: Trunc table** <tablename> ;

**Ex: trunc table** emp1 ;

**Destroying tables**

**Syntax: Drop table** <tablename> ;

**Ex: drop** table emp ;

**Data Manipulation Language (DML)** commands query and manipulate data in existing schema objects. These commands do not implicitly commit the current transaction.

Following are the commands:

1. Select
2. Insert
3. Delete
4. Update
5. Lock table
6. Explain Plan

**Inserting Data into Tables -** once a table is created the most natural thing to do is load this table with data to be manipulated later.

**Syntax: insert into** <tablename> (<col1>,<col2>) **values** (<exp>,<exp>) ;

**Delete operations**

**a)** remove all rows

**Syntax: delete from** <tablename> ;

**b)** removal of a specified row/s

**Syntax: delete from** <tablename> **where** <condition> ;

**Updating the contents of a table**

**a)** updating all rows

**Syntax: Update** <tablename> **set** <col> = <exp> , <col> = <exp> ;

**b)** updating selected records.

**Syntax:**

**Update**<tablename>**set** <col> = <exp> , <col> = <exp> where <condition>;

| Types of data constraints | Syntax |
|---|---|
| **not null** constraint at column level | <col> <datatype> (size) not null **;** |
| **unique** constraint at column level | <col> <datatype> (size) unique **;** |
| **unique** constraint at table level | Create table tablename (col = format, col = format, unique ( <col1>, <col2> ); |
| **primary key** constraint at column level | <col> <datatype> (size) primary key ; |
| **primary key** constraint at table level | Create table tablename (col = format, col = format primary key (col1>, <col2>) ; |
| **foreign key** constraint at **column level** | <col> <datatype>(size>)references<tablename> [<col>]; |
| **foreign key** constraint at **table level** | foreign key (<col>[,<col>]) references <tablename> [(<col>,<col>) ; |
| **Check constraintn** constraint at **column level** | <col> <datatype> (size) check (<logical expression>) ; |
| **Check constraint** constraint at **table level** | check (<logical expression>) ; |

**Transaction Control Commands** manages change made by Data Manipulation Language commands. Following are the commands:

1. **Commit**
2. **Rollback**
3. **Save point**
4. **Set Transaction**

Oracle provides extensive feature in order to safeguard information stored in its tables from unauthorized viewing and damage. The rights that allow the user of some or all oracle resources on the server are called privileges.

**Grant** privileges using the GRANT statement

The grant statement provides various types of access to database objects such as tables, views and sequences and so on.

**Syntax: GRANT** <object privileges> **ON** <objectname> **TO** <username>[WITH GRANT OPTION];

**REVOKE** statement:

The REVOKE statement is used to deny the Grant given on an object.

**Syntax:** REVOKE <object privilege> ON FROM <user name> ;

## Aggregate Functions

Aggregate functions return a single value based upon a set of other values. If used among many other expressions in the item list of a *SELECT* statement, the *SELECT* must have a *GROUP BY* clause. No *GROUP BY* clause is required if the aggregate function is the only value retrieved by the *SELECT* statement. The supported aggregate functions and their syntax are shown in following table.

| **Usage of Aggregate Functions** | **Function Name** |
|---|---|
| Computes the average value of a column by the expression | Avg() |
| Counts the rows defined by the expression | Count() |
| Counts all rows in the specified table or view | Count all() |
| Finds the minimum value in a column by the expression | Min() |
| Finds the maximum value in a column by the expression | Max() |
| Computes the sum of column values by the expression | Sum() |

**Syntax: Aggregate function name ( [ALL | DISTINCT] expression )**

The aggregate function name may be AVG, COUNT, MAX, MIN, or SUM. The ALL clause, which is the default behavior and does not actually need to be specified, evaluates all rows when aggregating the value of the function. The DISTINCT clause uses only distinct values when evaluating the function.

**AVG and SUM**

The AVG function computes the average of values in a column or an expression. SUM

computes the sum. Both functions work with numeric values and ignore NULL values. They also can be used to compute the average or sum of all distinct values of a column or expression.

AVG and SUM are supported by Microsoft SQL Server, MySQL, Oracle, and PostgreSQL.

**Explanation:-** The following query computes average year-to-date sales for each type of book:

```
SQL> SELECT   type, AVG( ytd_sales ) AS "average_ytd_sales"
       FROM   titles GROUP BY type;
```

This query returns the sum of year-to-date sales for each type of book:

```
SQL> SELECT   type, SUM ( ytd_sales )
       FROM  titles GROUP BY type;
```

**COUNT**

The COUNT function has three variations. COUNT (*) counts all the rows in the target table whether they include nulls or not. COUNT (expression) computes the number of rows with non-NULL values in a specific column or expression. COUNT (DISTINCT expression) computes the number of distinct non-NULL values in a column or expression.

**Explanation :-** This query counts all rows in a table:

**SQL>**SELECT COUNT (*) FROM publishers;

The following query finds the number of different countries where publishers are located:

**SQL>**SELECT COUNT (DISTINCT country) "Count of Countries" FROM publishers

**MIN and MAX**

MIN (expression) and MAX (expression) find the minimum and maximum value (string, date time, or numeric) in a set of rows. DISTINCT or ALL may be used with these functions, but they do not affect the result.

**Explanation :** The following query finds the best and worst sales for any title on record:

SELECT 'MIN' = MIN (ytd_sales), 'MAX' = MAX(ytd_sales) FROM titles;

Aggregate functions are used often in **the having clause** of queries with *GROUP BY*. The following query selects all categories (types) of books that have an average price for all books in the category higher than $15.00:

```
SQL> SELECT type 'Category', AVG( price ) 'Average Price'
     FROM titles
     GROUP BY type
     HAVING AVG(price) > 15
```

**CONCATENATE**

SQL99 defines a concatenation operator ( || ), which joins two distinct strings into one string value. The CONCATENATE function appends two or more strings together, producing a single output string. Oracle support the double-pipe concatenation operator. Microsoft SQL Server uses the plus sign (+) concatenation operator.

```
SQL> CONCATENATE ('string1' || 'string2')
```

## Practicing SQL Commands with examples

**Creating Tables**

In SQL*Plus we can execute any SQL command. One simple type of command creates a table (relation). The form is

**CREATE TABLE** <table Name> ( <list of attributes and their types> );

You may enter text on one line or on several lines. If your command runs over several lines, you will be prompted with line numbers until you type the semicolon that ends any command. (**Warning**: An empty line terminates the command but does not execute it; see editing commands in the buffer.) An example table-creation command is:

**CREATE TABLE** test ( i int, s char(10) );

Note that SQL is *case insensitive*, so CREATE TABLE TEST and create table test are the same. This command creates a table named test with two attributes. The first attribute, named i, is an integer, and the second, named s, is a character string of length (up to) 10.

**Exercise 1 : Create a relation Student that suitable for the following instance:**

| SID | NAME | JOB | SALARY | STREAM | START_AT |
|---|---|---|---|---|---|
| 1 | Ben Kao | Associate Professor | 7000 | E | 01-Sep-1995 |
| 2 | Eric Lo | Teaching Assistant | 1000 | E | 01-Oct-2003 |
| 3 | Hammer | Lecturer | 7000 | E | 11-Feb-2000 |
| 4 | Angela Castro | Program Manager | 6000 | I | 12-Dec-1999 |
| 5 | Steven Chu | Project Assistant | 7000 | I | 13-Dec-2002 |

**Note: No need to insert the data yet!**

**Inserting Tuples**

Having created a table, we can insert tuples into it. The simplest way to insert is with the INSERT command:

**INSERT INTO** <tableName> **VALUES** ( <list of values for attributes, in order> ) ;

For instance, we can insert the tuple (10, 'hi world') into relation test by

**Ex:** **INSERT INTO** test **VALUES** (10, 'hi world');

**Exercise 2: Insert the records as stated into Exercise 1 into the student table.**

**Trick**: Try to insert a record into test with the following SQL:

**INSERT INTO** test **VALUES** (11, 'ha 'world');

**Updating Tuples**

Tuples can be updated by the UPDATE command:

**UPDATE** <table Name> **SET** <Attribute> = <Expression / Value> **WHERE** <Predicate>;

For example, we can update the tuple (10, 'hi world') in relation test by

**Ex:** **UPDATE** test **SET** s='bye world' **WHERE** i=10;

**Exercise 3: Update the record of 'Eric Lo' in relation Student such that his salary change to 1234**

**UPDATE** student **set** salary = 1234 where sname = 'Eric Lo' ;

**Deleting Tuples**

Having insert / update a tuple, we can delete it as well. The simplest way to delete is with the DELETE command:

**DELETE FROM** <table Name> [**WHERE** <condition>];

<condition> is an optional statement and is used to identify a single record when necessary.

For example, you can delete the record with i=10 in table test with the the following SQL:

**Ex: DELETE FROM** test **WHERE** i=10;

**Exercise 4: Delete the record of 'Eric Lo' in relation Student.**

**Trick**: Does that record really deleted successfully? Let's check it out by using SELECT command (we will cover it in next section).

**Retrieving Tuples**

We can see the tuples in a relation with the command:

**SELECT** <attributes-separated-by-comma> **FROM** <tableName>;

For instance, after the above CREATE, INSERT DELETE and UPDATE statements, the command

SELECT * FROM test;

produces the result

I S
11 ha 'world

**Exercise 5: Select ALL records from relation Student.**

**Question**: Do data values also case insensitive? i.e., can a student with name "Hammer" be retrieved by the following SQL or not?

Select name from StudenT where name ='**hammer**';

**Commit and Rollback**

An automatic commit occurs under the following circumstance:

- DDL statement is issued
- Normal exit from SQL*Plus, without explicitly issuing COMMIT or ROLLBACK

An automatic rollback occurs under an abnormal termination of SQL*Plus or a system failure.

It provides a good back-door for you to revert the changes you have done on the data. Therefore, unless you have issued COMMIT, the changed data would not be visible to any other session except your own. Conversely, you can rollback all the changes by issuing the ROLLBACK command.

**Exercise 6: Issue the COMMIT command in the SQL*Plus that you have done insert/delete/update before, and see if the effect is now visible by the new SQL*Plus?**

**Dropping Tables**

To remove a table from your database, execute

**DROP TABLE** <table Name>;

We suggest you execute

**DROP TABLE** test;

Caution: Table dropping is a DML statement, which is an action that you cannot rollback. Since dropping a table will also delete all data in that table, issue the DROP TABLE command with cares.

**Getting Information about Your Database**

The system keeps information about your own database in certain system tables. The most important for now is USER_TABLES. You can recall the names of your tables by issuing the query:

SELECT TABLE_NAME FROM USER_TABLES;

More information about your tables is available from USER_TABLES. To see all the attributes of USER_TABLES, try:

SELECT * FROM USER_TABLES;

It is also possible to recall the attributes of a table once you know its name. Issue the command:

DESC <tableName>; to view the schema of <tableName>;

## Data Types

Here is part of the data types that are supported by Oracle.

| Data Type | Description |
|---|---|
| VARCHAR2 (*size*) | Variable-length character data (a maximum *size* must be specified: Minimum *size* is 1; maximum *size* is 4000) |
| CHAR [(*size*)] | Fixed-length character data of length *size* bytes (default and minimum *size* is 1; maximum *size* is 2000) |
| NUMBER [(*p*,*s*)] | Number having precision *p* and scale *s* (The precision is the total number of decimal digits, and the scale is the number of digits to the right of the decimal point; the precision can range from 1 to 38 and the scale can range from -84 to 127) |
| DATE | Date and time values to the nearest second between January 1, 4712 B.C., and December 31, 9999 A.D. |

**Creating Tables with Keys**

To create a table that declares attribute a to be a primary key:

CREATE TABLE <tableName> (..., a <type> PRIMARY KEY, b, ...);

To create a table that declares the set of attributes (a,b,c) to be a primary key:

CREATE TABLE <tableName> (<attrs and their types>, PRIMARY KEY (a,b,c));

## 6. List of Lab Exercises

| SL. NO | Name of the Program | PAGE NO. |
|---|---|---|
| I | **Product - Order System**<br><br>In recent years, most of the grocery items are available online; hence people are doing online transactions for purchase. There are lot of discounts and benefits through the online orders. Since everyone in the life is busy with one or other works, such applications will save their time.<br>These online transaction-based applications require many databases to be built for storage and transaction management. Design a product-order database which can store the details of customers, agents and the products. All the details of sold products along with commission from different agents across different cities will get stored in this database and utilized for transactions.<br><br>Customer (cid, cname, city, discount)<br>Agent (aid, aname, city, commission)<br>Product (pid, pname, city, quantity, price)<br>Orders (ordno, month, cid, aid, pid, qty, amount) | 23 |
| II | **Queries**<br><br>a. Retrieve the customer ids of any product which has been ordered by agent "a06".<br>b. Retrieve cities in which customers or agents located.<br>c. List product ids which have been ordered by agents from the cities “Dargeling” or “Srinagar”.<br>d. Retrieve customer ids whose discounts are less than the maximum discount.<br>e. Retrieve product ids ordered by at least two customers.<br>f. For each (aid, pid) pair get the sum of the orders aid has placed for pid.<br>g. Retrieve product ids and total quantity ordered for each product when the total exceeds 1000.<br>h. List the names of the customers and agent who placed an order through that agent.<br>i. Retrieve order numbers placed by customers in "Dargeling" through agents in "New Delhi".<br>j. Retrieve names of the customers who have the same discount as that of any (one) of the customers in "Dargeling" or "Bangalore".<br>k. Retrieve customer ids with smaller discounts than every customer from " Srinagar”<br>l. Retrieve names of the customers who have placed an order through agent "a05". (using exists )<br>m. Retrieve names of the customers who do not place orders through agent "a05". (using not exists)<br>n. Retrieve customer ids whose orders placed through all the agents in "New Delhi".<br>o. Retrieve agent ids either from "New Delhi" or "Srinagar" who place orders for ALL products priced over one dollar. | |

| | | |
|---|---|---|
| | p. Retrieve names and ids of the customers and agents along with total dollar sales for that pair. Order the result from largest to smallest total sales. Also retain only those pairs for which total dollar sales is at least 9000.00.<br>q. Increase the percent commission by 50% for all agents in "New York".<br>r. Retrieve the total quantity that has been placed for each product. | |
| **III** | **Employee Database System**<br><br>The storage of digital data is increasing day by day. Every big / small organization started storing their Employee details like name, salary, address, Department under which they are working in their own database. Design a company database which can store the details of Departments, projects, their Employee and his / her dependent details of a particular organization<br><br>Employee (ssn, name, salary, sex, super_ssn, address, dno)<br>Department (dname, dnumber,mgr_ssn)<br>Dept_Loc ( dnumber, dloc)<br>Project (pname, pnumber, plocation, dnum)<br>Works_On (essn, pno, hours)<br>Dependent (essn, depen_name, address, relationship, sex)<br>**Queries**<br><br>a. Retrieve the names of the Employees who works on all the projects controlled by dept no 3.<br>b. Retrieve the names of the Employees who gets second highest salary.<br>c. Retrieve the names of the Employees who have no dependents in alphabetical order.<br>d. List the names of all Employees with at least two dependents.<br>e. Retrieve the number of Employees and their average salary working in each Department.<br>f. Retrieve the highest salary paid in each Department in descending order.<br>g. Retrieve the SSN of all Employees who work on atleast one of the project numbers 1, 2, 3.<br>h. Retrieve the number of dependents for an Employee named RAM.<br>i. Retrieve the names of the managers working in location named xyz who has no female dependents.<br>j. Retrieve the names of the Employees who works in the same Department as that of RAM.<br>k. Retrieve the names of the Employees whose salary is greater than the salary of all the Employees working in Department no 3.<br>l. Retrieve the names of the Employees who work for dept no 3 and have a daughter as dependent.<br>m. Retrieve the names of the Employees who paid highest salary from each Department.<br>n. Retrieve the names of the Employees who are paid the same salary as that of Anil.<br>o. Retrieve the total the number of Employees in the ‘Research’ Department.<br><br>p. For each project, retrieve the project number, the project name, and the number of Employees who work on that project. | **41** |

# Lab Exercise 1: Product - Order database

| 1.1 | Problem Statement |
|---|---|

Design a product-order database which can store the details of customers, agents and the products. All the details of sold products along with commission from different agents across different cities will get stored in this database and utilized for transactions.

| 1.2 | Student Learning Outcomes |
|---|---|

After successful execution of this exercise, the student shall able to

- Design the database for all the real world applications.
- Implement aggregate functions.
- Apply set operations in database querying.

| 1.3 | Queries with Solutions |
|---|---|

Customer (cid, cname, city, discount)
Agent (aid, aname, city, commission)
Product (pid, pname, city, quantity, price)
Orders (ordno, month, cid, aid, pid, qty, amount)

**Aim**: Create the tables with the appropriate integrity constraints and Insert around 10 records in each of the tables

**SQL>** Create table Customer ( cid char(4) ,cname varchar(13) not null,city varchar(20),
discount real check(discount >= 0.0 and discount <= 15.0),
primary key (cid));

**Table created.**

**Explanation:** The above command will create a new table **Customer** in database system with 4 columns, namely cid, cname, city and discount using not null constraint for cname and primary key constraint for cid, discount checking with constraint as discount range+ greater than zero and less than 15 percent.

**SQL>** desc customer;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| CID | NOT NULL | CHAR(4) |
| CNAME | NOT NULL | VARCHAR2(13) |
| CITY | | VARCHAR2(20) |
| DISCOUNT | | FLOAT(63) |

**SQL>** Create table agent (aid char(3) ,
aname varchar(13) **not null**,
city varchar(20),
percent number(6) **check** (percent >= 0 and percent <= 100),
**primary key** (aid));

**Table created.**

**Explanation:** The above command will create a new table **agent** in database system with 4 columns, namely aid, aname, city and percent using not null constraint for aname and primary key constraint for aid, percent checking with constraint as percent range greater than or equal to zero and less than equal to 100 percent.

**SQL>** desc agent;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| AID | NOT NULL | CHAR(3) |
| ANAME | NOT NULL | VARCHAR2(13) |
| CITY | | VARCHAR2(20) |
| PERCENT | | NUMBER(6) |

**SQL>**Create table product ( pid char(3),pname varchar(13) unique not null,city varchar(20),
quantity number(10) check(quantity > 0),price real check(price > 0.0),primary key (pid));

**Table created.**

**Explanation:** The above command will create a new table **Product** in database system with 5 columns, namely pid, pname, city , Quantity and price using not null constraint for pname and primary key constraint for pid , price checking with constraint as range greater than zero.

**SQL>** desc product;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| PID | NOT NULL | CHAR(3) |
| PNAME | NOT NULL | VARCHAR2(13) |
| CITY | | VARCHAR2(20) |
| QUANTITY | | NUMBER(10) |
| PRICE | | FLOAT(63) |

create table order (
*
ERROR at line 1:
ORA-00903: invalid table name
As order is a reserved word in oracle we cant create a table with name "order" so used orders

```
SQL>  Create table orders ( ordno number(6),
                            month char(3),
                            cid char(4) not null,
                            aid char(3) not null,
                            pid char(3) not null,
                            qty number(6) not null check(qty > 0),
                            ordamount float  default 0.0 check(ordamount >= 0.0),
                            primary key (ordno),
                            foreign key (cid) references customer,
                            foreign key (aid) references agent,
                            foreign key (pid) references product);
```

**Table created.**

**Explanation:** The above command will create a new table **orders** in database system with 7 columns, namely ordno, month, cid ,aid, pid, Qty and ordamount and with foreign key constraint of cid referring to customer and aid referring agent, pid referring product, ordno as primary key and quantity with check constraint with range greater than zero .

**SQL>** desc order;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| ORDNO | NOT NULL | NUMBER(6) |
| MONTH | | CHAR(3) |
| CID | NOT NULL | CHAR(4) |
| AID | NOT NULL | CHAR(3) |
| PID | NOT NULL | CHAR(3) |
| QTY | NOT NULL | NUMBER(6) |
| DOLLARS | | FLOAT(126) |

**Insert the data into the table customer, agent , product, orders**

**SQL>** insert into customer values (’c001’,’Sobhit’,’Darjeling’,10.00);
1 row created.
**SQL>**insert into customer values (’c002’,’Bhanu’,’Srinagar’,12.00);
1 row created.
**SQL>**insert into customer values (’c003’,’Amar’,’ Srinagar’,8.00);
1 row created.
**SQL>**insert into customer values (’c004’,’Anand’,’Darjeling’,8.00);
1 row created.
**SQL>**insert into customer values (’c005’,’Anand’,’Mumbai’,0.00);
1 row created.

**NOTE:** If an attempt is made to insert the same cid , as it is having primary constraint it shows an error.

**SQL>** insert into customer values ('c001','Sachin','Darjeling',10.00);

1 row created.

**SQL**>insert into customer values ('c001','Sachin','Darjeling',10.00)*

ERROR at line 1:
ORA-00001: unique constraint (DBMS.SYS_C005054) violated

**SQL> select * from customer;**

**OUTPUT:**

| CID | CNAME | CITY | DISCOUNT |
|---|---|---|---|
| c001 | Sobhit | Darjeling | 10 |
| c002 | Bhanu | Srinagar | 12 |
| c003 | Amar | Srinagar | 8 |
| c004 | Anand | Darjeling | 8 |
| c005 | Anand | Mumbai | 0 |

**SQL**>insert into agent values('a01','Sonu','NewDelhi',6.00);
1 row created.
**SQL**>insert into agent values('a02','John','Agra',6.00);
1 row created.
**SQL**>insert into agent values('a03','Bhargav','Jaipur',7.00);
1 row created.
**SQL**>insert into agent values('a04','Gaurav','NewDelhi',6.00);
1 row created.
**SQL**>insert into agent values('a05','Omkar','Srinagar',5.00);
1 row created.
**SQL**>insert into agent values('a06','Sonu','Darjeling',5.00);
1 row created.

**SQL**> select * from agent;

**OUTPUT:**

| AID | ANAME | CITY | PERCENT |
|---|---|---|---|
| a01 | Sonu | NewDelhi | 6 |
| a02 | John | Agra | 6 |
| a03 | Bhargav | Jaipur | 7 |
| a04 | Gaurav | NewDelhi | 6 |
| a05 | Omkar | Srinagar | 5 |
| a06 | Sonu | Darjeling | 5 |

6 rows selected.

**SQL**>insert into product values('&PID','&PNAME','&CITY','&QUANTITY','&PRICE');

**OUTPUT:**

| PID | PNAME | CITY | QUANTITY | PRICE |
|---|---|---|---|---|
| p01 | comb | Darjeling | 100000 | 10 |
| p02 | brush | Agra | 200000 | 20 |
| p03 | eraser | Srinagar | 150000 | 2 |
| p04 | pen | Srinagar | 100000 | 15 |
| p05 | pencil | Darjeling | 170000 | 3 |
| p06 | folder | Darjeling | 180000 | 15 |
| p07 | Highlighter | Agra | 180000 | 20 |

**SQL**> insert into orders values ('&ordno','&month','&cid','&aid','&pid',&qty,&ordamount);

```
Enter value for ordno:    1011
Enter value for month:     jan
Enter value for cid:        c001
Enter value for aid:        a01
Enter value for pid:       p01
Enter value for qty:       1000
Enter value for ordamount: 9400

old   1: insert into orders values ('&ordno','&month','&cid','&aid','&pid',&qty,&ordamount)
new   1: insert into orders values ('1011','jan','c001','a01','p01',1000,9400)
1 row created.
```

**SQL> /**

```
Enter value for ordno:  1012
Enter value for month:  jan
Enter value for cid:       c001
Enter value for aid:       a01
Enter value for pid:       p01
Enter value for qty:      1000
Enter value for ordamount: 9400
```

**SQL>** select * from orders;

**OUTPUT:**

| ORDNO | MON | CID | AID | PID | QTY | ORDAMOUNT |
|---|---|---|---|---|---|---|
| 1011 | jan | c001 | a01 | p01 | 1000 | 9400 |
| 1012 | jan | c001 | a01 | p01 | 1000 | 9400 |
| 1013 | jan | c002 | a03 | p03 | 1000 | 1860 |
| 1014 | jan | c003 | a03 | p05 | 1200 | 3348 |
| 1015 | jan | c003 | a03 | p05 | 1200 | 3348 |
| 1016 | jan | c005 | a01 | p01 | 1000 | 9400 |
| 1017 | feb | c001 | a02 | p02 | 400 | 7520 |
| 1018 | feb | c001 | a03 | p04 | 600 | 2232 |
| 1019 | feb | c001 | a02 | p02 | 400 | 7520 |
| 1020 | feb | c005 | a03 | p07 | 600 | 11160 |
| 1021 | feb | c004 | a06 | p01 | 1000 | 9500 |
| 1022 | mar | c001 | a05 | p06 | 400 | 5700 |
| 1023 | mar | c001 | a04 | p05 | 500 | 1410 |
| 1024 | mar | c005 | a06 | p01 | 800 | 7600 |
| 1025 | Apr | c001 | a05 | p07 | 800 | 15200 |

15 ows selected.

**2.a ) Retrieve the customer ids of any product which has been ordered by agent "a06".**

**SQL>** select distinct p.cid from orders o, orders p where p.pid=o.pid and o.aid='a06'

**Explanation:** Distinct keyword gives the different values of attribute cid from table Orders and product with the join ,pid attribute from product table and pid attribute from order table, whose agent id is ao6.

**OUTPUT:**

| CID |
|---|
| c001 |
| c004 |
| c005 |

**2.b) Retrieve cities in which customers or agents located.**

**SQL>**
```
select city from customer
      union
select city from agent;
```

**Explanation:** This query retrieves the city names as the union operator helps to combine both the tables customer and agent containing column name city and it won't allow duplicate values.

**OUTPUT:**

| CITY |
|---|
| Agra |
| Darjeling |
| Jaipur |
| Mumbai |
| NewDelhi |
| Srinagar |

6 rows selected

**2.c )List product ids which have been ordered by agents from the cities "Dargeling" or "Srinagar".**

**SQL>**
```
select  distinct(o.pid) from orders o ,agent a  where  o.aid=a.aid and
      a.city in( 'Darjeling','Srinagar');
```

**Explanation:** Distict helps to select the distinct values of pid attribute from order table and agent table, and the columns aid from tables order and agents with city 'Darjeeling' and Srinagar using in operator.

Or

**SQL>**
```
select  distinct(pid) from orders where  aid in (select aid from agent where
      city in( 'Darjeling','Srinagar'));
```

**Explanation:** Distinct selects the distinct values of pid from orders table and using in operator to select the aid attribute from agent table with city names darjeling and Srinagar.

**OUTPUT:**

| PID |
|---|

p01
p06
p07

**2.d) Retrieve customer ids whose discounts are less than the maximum discount.**

**SQL>** select cid from customer
where discount < ( select max(discount) from customer);

**Explanation:** This query gives the customer ids from customer table with condition whose discount is less than max discount as max(discount) gives the maximum discount from customer table.

**OUTPUT:**

| CID |
|---|
| c001 |
| c003 |
| c004 |
| c005 |

**2.e ) Retrieve product ids ordered by at least two customers.**

**SQL>** select p.pid from product p
where 2 < = (select count(distinct cid) from orders
where pid = p.pid);

**Explanation:** This Query gives the pids from prouct table ,with condition 2<= count(distinct cid) as count gives the no.of distinct cids from orders table and pid column from product table .

**OUTPUT:**

| PID |
|---|
| p01 |
| p05 |
| p07 |

**2.f) For each (aid,pid) pair get the sum of the orders aid has placed for pid.**

**SQL>** select pid, aid, sum(qty) TOTAL
from orders
group by pid, aid;

**Explanation:** To retrieve pid and aid attribute and the sum operator is used to return the total sum of the qty column from orders table using group by function is used to get the result in a set of pid and aid attributes.

**OUTPUT:**

| PID | AID | TOTAL |
|---|---|---|
| p01 | a01 | 3000 |

| | | |
|---|---|---|
| p01 | a06 | 1800 |
| p02 | a02 | 800 |
| p03 | a03 | 1000 |
| p04 | a03 | 600 |
| p05 | a03 | 2400 |
| p05 | a04 | 500 |
| p06 | a05 | 400 |
| p07 | a03 | 600 |
| p07 | a05 | 800 |

10 rows selected.

**2.g ) Retrieve product ids and total quantity ordered for each product when the total exceeds 1000.**

```
SQL>    select pid, aid, sum(qty) TOTAL
            from orders
                group by pid, aid
                  having suzm(qty) > 1000;
```

**Explanation:** To retrieve pid and aid attributes and the sum operator is used to return the total sum of the qty from orders table using group by function is used to get the result in a set of pid and aid attributes and having clause is used instead of where as condition whose sum(qty) is greater than 1000.

**OUTPUT:**

| PID | AID | TOTAL |
|---|---|---|
| p01 | a01 | 3000 |
| p01 | a06 | 1800 |
| p05 | a03 | 2400 |

**2.h) List the names of the customers and agent who placed an order through that agent.**

```
SQL>    select distinct cname, aname
            from customer, orders, agent
                where customer.cid = orders.cid and
                      orders.aid = agent.aid;
```

**Explanation:** This query gives distinct values from attributes v=cname,aname from customer, orders and agent tables with condition cid from customer table and orders and also from aid attribute from orders and agent table.

**OUTPUT:**

| CNAME | ANAME |
|---|---|
| Amar | Bhargav |
| Anand | Bhargav |

| | |
|---|---|
| Anand | Sonu |
| Bhanu | Bhargav |
| Sobhit | Bhargav |
| Sobhit | Gaurav |
| Sobhit | John |
| Sobhit | Omkar |
| Sobhit | Sonu |

6 rows selected.

**2 i) Retrieve the order numbers placed by customers in "Dargeling" through agents in "NewDelhi".**

**SQL>**select ordno from orders where cid in (select cid from customer where city = 'Darjeling') and aid in (select aid from agent where city = 'NewDelhi');

**Explanation:** To get ordno from orders table with condition using in operator for attribute cid from customer table whose city is darjeling and aid from agent table whose city is newdelhi.

**OUTPUT:**

| ORDNO |
|---|
| 1011 |
| 1012 |
| 1023 |

**2 j) Retrieve names of the customers who have the same discount as that of any (one) of the customers in "Dargeling" or "Bangalore".**

```
SQL>        select cname from customer
                where discount =any (select discount from customer
                                     where city = 'Darjeling' or city = 'Bangalore');
```

**Explanation:** To get the cname from customer table with condition where any value of discount from customer table whose city is Darjeeling or Bangalore.

**OUTPUT:**

| CNAME |
|---|
| Sobhit |
| Anand |
| Amar |

**2 b K) Retrieve customer ids with smaller discounts than every customer from " Srinagar"**

```
SQL>        select cid from customer
                where discount < all (select discount from customer
                                      where city = 'Srinagar');
```

**Explanation:** To get cid from customer table, with condition whose all discount values is less than the discount of every customer whose city is Srinagar.

**OUTPUT:** CID
c005

**2 B I) Retrieve names of the customers who have placed an order through agent "a05"**
**(using exists )**

**SQL>**
```
select c.cname from customer c
    where exists (select * from orders o
                  where c.cid = o.cid and o.aid = 'a05');
```

**Explanation:** To get the cname from customer table as exist helps to check the existence of query and selects the complete table from orders with condition cid attribute from customer table and order table whose aid column from order table is a05.

or

**SQL>**
```
select cname from customer  where cid in (select cid from orders where aid='a05');
```

**OUTPUT:** CNAME
Sobhit

**2 B m) Retrieve names of the customers who do not place orders through agent "a05".**
**(using not exists)**

**SQL>**
```
select cname from customer
    where cid not in  (select cid from orders where orders.aid = 'a05');
```

**Explanation:** To retrieve cname from customer table with condition cid from orders table is not in cid of order table where aid from order table is a05

or

**SQL>**
```
select cname from customer
    where cid <>any (select cid from orders where orders.aid = 'a05');
```

**OUTPUT:** CNAME
Bhanu
Amar
Anand
Anand

**2 B n) Retrieve customer ids whose orders placed through all the agents in "New Delhi".**

**Get cid values of customers such that (the set of agents from " NewDelhi " through whom the customer has NOT placed an order) is EMPTY.**

**SQL>**
```
select c.cid  from customer c
        where not exists  (select * from agent a where a.city = 'NewDelhi'
                                and
                not exists (select * from orders o,customer c ,agent a where
                                o.cid=c.cid and o.aid=a.aid));
```

**Explanation:** To retrieve cid from customer table ,with condition not exists,as this helps to get the values which are not existed in the agent table whose city is new delhi and also from orders table that do not exist with condition cid column from order table equal to cid column from customer table and also aid columns from order and agent tables.

**OUTPUT:**

| CID |
|---|
| c00 |

**2 b o). Retrieve agent ids either from "NewDelhi" or "Srinagar" who place orders for ALL products priced over fifteen rupee. Get aid values of agents from "New York" or "Duluth" such that (the set of products priced over one dollar that the agent has NOT ordered) is EMPTY.**

**SQL>**
```
select a.aid from agent a
      where (a.city in ('NewDelhi','Srinagar')) and
            not exists (select p.pid from product p where p.price > 15.00
                  and
                  not exists (select * from orders o
                        where o.pid = p.pid and o.aid = a.aid));
```

**Explanation:**To retrieve aid from agent table whose cities are new delhi and srinagar and using not exist to select the pid from product whose price is greater than 15,and also again using not exist from orders table where pid from order and product table also aid from both tables are equal.

**OUTPUT:**

no rows selected

So

```
insert into orders values('1026','apr','c005','a05','p02',900,17100);
```

**OUTPUT:**

| AID |
|---|
| a05 |

**2 b p).Retrieve names and ids of the customers and agents along with total sales for that**

**pair. Order the result from largest to smallest total sales. Also retain only those pairs for which total rupee sales is at least 9000.00.**

```
SQL>  select c.cname, c.cid, a.aname, a.aid, sum(o.ordamount)
          from customer c, orders o, agent a
              where c.cid = o.cid and o.aid = a.aid
                  group by c.cname, c.cid, a.aname, a.aid
                      having sum(o.ordamount) >= 9000.00
                          order by 5 desc;
```

**Explanation:**To retrieve cname,cid,from customer table and aname and aid from agent table,with sum function for order amount from order table ,customer and agent table with condition cid from customer and order tables and aid from order and agent tables are equal and to get the result in one set group by is used for cname,cid of customer table and aid ,aname from agent table with sum of ordramount of order table is >= 9000 in descending order as result.

**OUTPUT:**

| CNAME | CID | ANAME | AID | SUM(O.ORDAMOUNT) |
|---|---|---|---|---|
| Sobhit | c001 | Omkar | a05 | 20900 |
| Sobhit | c001 | Sonu | a01 | 18800 |
| Sobhit | c001 | John | a02 | 15040 |
| Anand | c005 | Bhargav | a03 | 11160 |
| Anand | c004 | Sonu | a06 | 9500 |
| Anand | c005 | Sonu | a01 | 9400 |

6 rows selected.

**2 B q) Increase the percent commission by 50% for all agents in "NewDelhi".**

```
SQL>  update agent
          set percent = 1.5 * percent
              where city = 'NewDelhi;
```

**Explanation:**To update agent table, percentage value is set to 1.5*percent to get 50% whose city is NewDelhi.

**2 B r). Retrieve the total quantity that has been placed for each product**

```
SQL>  select pid, sum(qty) TOTAL  from orders  group by pid;
```

**Explanation:** To get pid with total sum of qty from orders table using group by to get the result in one set.

**OUTPUT:** PID TOTAL

| | |
|---|---|
| p01 | 4800 |
| p02 | 800 |
| p03 | 1000 |
| p04 | 600 |
| p05 | 2900 |
| p06 | 400 |
| p07 | 1400 |

3 rows selected.

## Lab Exercise 3 : Employee Database System

| 3.1 | **Problem Statement** |
|---|---|
| Designing a company database which can store Department, project, Employee and his dependent details of a particular organization**.** | |
| **3.2** | **Student Learning Outcomes** |

After successful execution of this exercise, the student shall able to

- Design the database for all the bknreal-world applications.
- Implement group by clause functions.
- Apply any, all operations in database querying.

**3.3 Queries with Solutions**

Employee (ssn, name, salary, sex, super_ssn, address, dno)
Department (dname, dnumber)
Dept_Loc ( dnumber, dloc, mgrssn)
Project (pname, pnumber, plocation, dnum)
Works_On (essn, pno, hours)
Dependent (essn, depen_name, address, relationship, sex)

**Aim**: Create the tables with the appropriate integrity constraints and Insert around 10 records in each of the tables

**NOTE:** Department table has a column mgr_ssn which is a foreign key referring to ssn column of an Employee table And Employee table has a column dno which is a foreign key referring to dnumber of Department table.So it is interlinked and deadlock appears.

step i create table Department with attributes dno, dname (without mgr_ssn column)
step ii insert data into Department
step iii create table Employee and insert data ( without super_ssn column)
step iv insert data into Employee table
step v add new column super_ssn into Employee table and update data to column super_ssn
step vi add new column mgrssn into Department referring to Employee table
stpe vii insert data of mgrssn in Department table

**SQL**>Create table Department ( dname varchar(15),
unique **not null**,
dnumber int ,
**primary key** (dnumber));

**Table Created.**

**SQL**> desc Department;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| DNAME | NOT NULL | VARCHAR2(15) |
| DNUMBER | NOT NULL | NUMBER(38) |

**SQL>** Insert into Department values ('Research',1);
1 row created.
**SQL>** Insert into Department values ('HR',2);
1 row created.
**SQL>** Insert into Department values ('Development',3);
1 row created.
**SQL>** Insert into Department values ('Testing',4);
1 row created.
**S**t * from Department;

**OUTPUT:**

| DNAME | DNUMBER |
|---|---|
| Research | 1 |
| HR | 2 |
| Development | 3 |
| Testing | 4 |

**SQL>** Create table Employee ( ssn char(9),
name varchar(15) **not null**,
salary decimal(10,2),
sex char,
address varchar(30),
dno int **not null**,
**primary key**(ssn),
**foreign key**(dno) **references** Department(dnumber));

**Table Created.**

**SQL>** desc Employee;

**OUTPUT :**

| **Name** | **Null?** | **Type** |
|---|---|---|
| SSN | NOT NULL | CHAR(9) |
| NAME | NOT NULL | VARCHAR2(15) |
| SALARY | | NUMBER(10,2) |
| SEX | | CHAR(1) |
| ADDRESS | | VARCHAR2(30) |
| DNO | NOT NULL | NUMBER(38) |

**SQL>**Insert into Employee values('emp001','Ram',30000,'M','RT Nagar, Blore',3);
**SQL>**Insert into Employee values('emp002','Sudha',75000,'F','Hebbal, Blore',2);
**SQL>**Insert into Employee values('emp003','Ravi',20000,'M','Hebbal, Blore',4);
**SQL>**Insert into Employee values('emp004','Rohan',80000,'M','RT Nagar, Mysore',1);
**SQL>**Insert into Employee values('emp005','Amar',35000,'M','MG Road, Mysore',3);
**SQL>**Insert into Employee values('emp006','Anil',45000,'M','MG Road, Noida',3);
**SQL>**Insert into Employee values('emp007','Tanya',35000,'F','Yelahanka, Blore',3);
**SQL>**Insert into Employee values('emp008','Kavita',50000,'F','Baglur, Blore',1);

**SQL>**Insert into Employee values('emp009','John',45000,'M','RT Nagar, Blore',4);

**SQL>** select * from Employee;

**OUTPUT:**

| SSN | NAME | SALARY | SEX | ADDRESS | DNO |
|---|---|---|---|---|---|
| emp001 | Ram | 30000 | M | RT Nagar, Blore | 3 |
| emp002 | Sudha | 75000 | F | Hebbal, Blore | 2 |
| emp003 | Ravi | 20000 | M | Hebbal, Blore | 4 |
| emp004 | Rohan | 80000 | M | RT Nagar, Mysore | 1 |
| emp005 | Amar | 35000 | M | MG Road, Mysore | 3 |
| emp006 | Anil | 45000 | M | MG Road, Noida | 3 |
| emp007 | Tanya | 35000 | F | Yelahanka, Blore | 3 |
| emp008 | Kavita | 50000 | F | Baglur, Blore | 1 |
| emp009 | John | 45000 | M | RT Nagar, Blore | 4 |

**SQL> alter table** Employee **add** super_ssn char(9) **references** Employee(ssn);
Table altered.

**SQL>**update Employee set super_ssn=’emp006’ where ssn=’emp001’;
**SQL>**update Employee set super_ssn=’emp008’ where ssn=’emp003’;
**SQL>**update Employee set super_ssn=’emp002’ where ssn=’emp005’;
**SQL>**update Employee set super_ssn=’emp008’ where ssn=’emp006’;
**SQL>**update Employee set super_ssn=’emp008’ where ssn=’emp007’;
**SQL>**update Employee set super_ssn=’emp004’ where ssn=’emp008’;
**SQL>**update Employee set super_ssn=’emp008’ where ssn=’emp009’;

**SQL>** select * from Employee**;**

**OUTPUT:**

| SSN | NAME | SALARY | S | ADDRESS | DNO | SUPER_SSN |
|---|---|---|---|---|---|---|
| emp001 | Ram | 30000 | M | RT Nagar, Blore | 3 | emp006 |
| emp002 | Sudha | 75000 | F | Hebbal, Blore | 2 | |
| emp003 | Ravi | 20000 | M | Hebbal, Blore | 4 | emp008 |
| emp004 | Rohan | 80000 | M | RT Nagar, Mysore | 1 | |
| emp005 | Amar | 35000 | M | MG Road, Mysore | 3 | emp002 |
| emp006 | Anil | 45000 | M | MG Road, Noida | 3 | emp008 |
| emp007 | Tanya | 35000 | F | Yelahanka, Blore | 3 | emp008 |
| emp008 | Kavita | 50000 | F | Baglur, Blore | 1 | emp004 |
| emp009 | John | 45000 | M | RT Nagar, Blore | 4 | emp008 |

**SQL> alter table** Department **add** mgr_ssn char(9) **references** Employee(ssn);

**SQL>** desc Department;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| DNAME | NOT NULL | VARCHAR2(15) |
| DNUMBER | NOT NULL | NUMBER(38) |
| MGR_SSN | | CHAR(9) |

**SQL**> select * from Department;

**OUTPUT:**

| DNAME | DNUMBER | MGR_SSN |
|---|---|---|
| Research | 1 | |
| HR | 2 | |
| Development | 3 | |
| Testing | 4 | |

**SQL**>update Department set mgr_ssn='emp004' where dnumber=1;
**SQL**>update Department set mgr_ssn='emp002' where dnumber=2;
**SQL**>update Department set mgr_ssn='emp006' where dnumber=3;
**SQL**>update Department set mgr_ssn='emp009' where dnumber=4;

**SQL**> select * from Department;

**OUTPUT:**

| DNAME | DNUMBER | MGR_SSN |
|---|---|---|
| Research | 1 | emp004 |
| HR | 2 | emp002 |
| Development | 3 | emp006 |
| Testing | 4 | emp009 |

**SQL**>Create table Dept_Location ( dnumber int **not null**,
dlocation varchar(15) **not null**,
**primary key**(dnumber, dlocation),
**foreign key**(dnumber) **references** Department(dnumber) );

**Table Created.**

**SQL**> desc Dept_Location;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| DNUMBER | NOT NULL | NUMBER(38) |
| DLOCATION | NOT NULL | VARCHAR2(15) |

**SQL**> Create table Project ( pname varchar(15) **not null**,
pnumber varchar(5) **not null,**
plocation varchar(15),
dnum int **not null**,

**primary key** (pnumber),
**unique**(pname),
**foreign key** (dnum) **references** Department(dnumber));

**Table Created.**

**SQL>** desc Project;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| PNAME | NOT NULL | VARCHAR2(15) |
| PNUMBER | NOT NULL | VARCHAR2(5) |
| PLOCATION | | VARCHAR2(15) |
| DNUM | NOT NULL | NUMBER(38) |

**SQL>**Create table Workson ( essn char(9) **not null**,
pno varchar(5) **not null**,
hours decimal(3,1) **not null** ,
**primary key**(essn, pno),
**foreign key** (essn) **references** Employee(ssn),
**foreign key** (pno) **references** project(pnumber));

**Table Created.**

**SQL>** desc Workson;

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| ESSN | NOT NULL | CHAR(9) |
| PNO | NOT NULL | VARCHAR2(5) |
| HOURS | NOT NULL | NUMBER(3,1) |

**SQL>** Create table Dependent ( essn char(9) **not null**,
dependent_name varchar(15) **not null**,
sex char,
relationship varchar(8),
**primary key** (essn,),
**foreign key** (essn) **references** Employee(ssn));

**Table Created.**
**SQL> desc Dependent;**

**OUTPUT:**

| Name | Null? | Type |
|---|---|---|
| ESSN | NOT NULL | CHAR(9) |
| DEPENDENT_NAME | NOT NULL | VARCHAR2(15) |
| SEX | | CHAR(1) |
| RELATIONSHIP | | VARCHAR2(8) |

**SQL>** select * from Dept_Location;

**OUTPUT**:

| DNUMBER | DLOCATION |
|---|---|
| 1 | Blore |

| | |
|---|---|
| 2 | Blore |
| 3 | Blore |
| 3 | Mysore |
| 4 | Noida |
| 4 | Blore |

**SQL>** select * from Project;

**OUTPUT:**

| PNAME | PNUMBER | PLOCATION | DNUM |
|---|---|---|---|
| Banking | p01 | Blore | 3 |
| Android App | p02 | Mysore | 3 |
| WSN | p03 | Blore | 4 |
| Robotics | p04 | Noida | 4 |
| Smart Vehicle | p05 | Blore | 3 |

**SQL>** select * from Workson;

**OUTPUT:**

| ESSN | PNO | HOURS |
|---|---|---|
| emp001 | p01 | 14 |
| emp003 | p01 | 10 |
| emp001 | p02 | 7 |
| emp005 | p03 | 18 |
| emp003 | p02 | 14 |
| emp004 | p05 | 12 |
| emp007 | p04 | 14 |
| emp001 | p05 | 12 |

**SQL**> select * from Dependent;

**OUTPUT:**

| ESSN | DEPENDENTNAME | SEX | RELATION |
|---|---|---|---|
| emp001 | Raghu | M | son |
| emp004 | Reshma | F | wife |
| emp007 | Bindu | F | daughter |
| emp009 | Shaan | M | son |
| emp009 | Shamir | M | son |

**a. Retrieve the names of the Employees who gets second highest salary.**

**SQL>** select name from Employee
where salary in ( select max(salary) from Employee
where salary not in (select max(salary) from Employee));

or

**SQL>** select name from Employee
where salary in ( select max(salary) from Employee
where salary < (select max(salary) from Employee));

**Explanation**: This Query gives the name of the Employee using max function for salary column from Employee table, whose sal is not in max(salary) or by using < operator .

**OUTPUT:** NAME
Sudha

**b. Retrieve the names of the Employees who have no dependents in alphabetical order.**

**SQL>** select name from Employee e
where not exists (select * from dependent where essn=e.ssn) order by name;
or

**SQL>** select name from Employee
where ssn not in (select essn from dependent) order by name;

**Explanation:** This query gives the name of Employee using not exists in dependent with condition essn=e.ssn using order by name. or

By using not in and ssn column from dependent table using order by .

**OUTPUT:** NAME
Anil
Sudha
Ravi
Amar
Kavita

**c. List the names of all Employees who have at least two dependents**

**SQL>** select name from Employee
where ( select count (*) from Dependent where ssn =Essn )>=2;

**Explanation:** This query gives the names of the Employees, using count function as it gives the count value of the selected table with condition ssn=Essn and >=2.

**OUTPUT :** NAME
John

**d. Retrieve the number of Employees and their average salary working in each department.**

**SQL>** select dno, count (*),avg(salary) from Employee group by dno;

**Explanation:** This Query gives dno,count and avg sal from Employee table using group by dno .

**OUTPUT :**

| DNO | COUNT(*) | AVG(SALARY) |
|---|---|---|
| 1 | 2 | 65000 |
| 2 | 1 | 75000 |
| 3 | 4 | 36250 |
| 4 | 2 | 32500 |

**e. Retrieve the highest salary paid in each department in descending order.**

**SQL>** select dno,max(salary) from Employee group by dno order by max(salary) desc;

**Explanation:** This Query gives the dno and Max (salary) Using Group by and Oder by functions with conditions group by dno and order by max(salary) as desc.

**OUTPUT :**

| DNO | MAX(SALARY) |
|---|---|
| 1 | 80000 |
| 2 | 75000 |
| 3 | 45000 |
| 4 | 45000 |

**f. Retrieve the SSN of all Employees who work on atleast one of the project numbers 1, 2, 3**

**SQL>** select distinct(essn) from workson where pno in ('p01','p02','p03');

or

**SQL>** select distinct(essn) from workson where pno='p01'or pno='p02' or pno='p03';

**Explanation** : This Query gives the Essn of Employee from workson table using distinct with condition pno in (p01,p02,p03) or pno=p01 or p02,p03

**OUTPUT :**

| ESSN |
|---|
| emp001 |
| emp003 |
| emp005 |

**g. Retrieve the number of dependents for an Employee named RAM.**

**SQL>** select count(*) from Dependent
where essn = (select ssn from Employee where name='Ram');

or

**SQL>** select count(*) from Employee e, dependent d
where d.essn=e.ssn and e.name='Ram';

**Explanation** : This Query gives the no .of dependents using count function for dependent table with condition whose essn is equal to ssn of Employee table whose ename is ram .

Or

**Explanation:** By using count for Employee table and dependent with condition essn of dependent table = ssn of Employee table and whose name from Employee table is ram.

**OUTPUT :**

| COUNT(*) |
|---|
| 1 |

**h. Retrieve the names of the managers working in location named xyz who has no female dependents.**

```
SQL> select name from Employee
        where ssn in ( select essn from dependent
                        where sex!='F'
                            and essn in(
                                (select mgr_ssn from department
                                    where dnumber in
                                (select dnumber from dept_location where dlocation='Blore'))));
```

Or

```
SQL>  select distinct(name) from Employee e , Department d, Dept_location l, Dependent de
            where e.ssn= de.essn and de.sex!='F'  and
                        de.essn=d.mgr_ssn and
                        d.dnumber=l.dnumber  and
                        l.dlocation='Blore';
```

**Explanation:** This query gives the name of manger from Employee table by using IN operator, whose ssn from dependent table sex is not 'F' and mgr_ssn from department,whose dnumber from dep_location is Blore.

Or

**Explanation:** By using distinct for name from Employee table ,department ,dependent and location tables whose ssn from Employee =essn from dependent table and sex is not 'F' in dependent and essn from dependent =mgr from department mgr_ssn and dnumber from d=dhumber from l;and dlocation from l='Blore'.

**OUTPUT :**

| NAME |
|---|
| John |

**i. Retrieve the names of the Employees who works in the same department as that of**

**RAM**

**SQL>** select name from Employee
where dno = (select dno from Employee where name='Ram') and name!='Ram';

**Explanation:** This gives the name from Employee using condition get dno whose name is ram from emp table and name whose name is not ram.

**OUTPUT :**

| NAME |
|---|
| Amar |
| Anil |
| Tanya |

**j. Retrieve the name of the Employees whose salary is greater than the salary of all the Employees working in department 3.**

**SQL>** select name, salary from Employee
where salary> all ( select salary from Employee where dno=3 );

**Explanation:** To get the name, salary of Employee from emp table using condition whose salary is > All Employees and whose dno=3.

**OUTPUT :**

| NAME | SALARY |
|---|---|
| Sudha | 75000 |
| Rohan | 80000 |
| Kavita | 50000 |

**k. Retrieve the names of the Employees who work for dept no 3 and have a daughter as dependent.**

**SQL>** select name from Employee e , dependent d
where e.ssn=d.essn and d.relationship='daughter' and e.dno=3;

**Explanation:** This Query gives the name using conditions ssn fromemp=essn from dependent and relationship from depend=daughter and dno from emp=3.

**OUTPUT :**

| NAME |
|---|
| Tanya |

**l. Retrieve the Employee name who paid highest salary from each department.**

**SQL>** select name from Employee
where salary in ( select max(salary) from Employee group by dno ) ;

**Explanation:** To get the name from emp using condition whose sal in emp using max and groupby dno.

**OUTPUT :**

| NAME |
|---|
| Rohan |
| Sudha |
| John |
| Anil |

**m. Retrieve the names of the Employees who are paid the same salary as that of Anil.**

```
SQL>    select name from Employee
            where salary in (select salary from Employee where name='Anil')
                and     name!='Anil';
        or

SQL>    select name from Employee
            where salary = (select salary from Employee where name='Anil')
                and   name!='Anil';
```

**Explanation** This gives the name of Employee using In operator and condition sal frm emp table whose name is Anil and name frm emp whose name is not anil.

**OUTPUT :**

| NAME |
|---|
| John |

**n. Retrieve the total the number of Employees in the 'Research' department.**

```
SQL>  select count (*) from Employee, Department
          where dno =dnumber and  dname='Research';
```

**Explanation:** To get the total ,we use count function along with condition whose dno is dnumber and dname is research.

**OUTPUT :**

| COUNT(*) |
|---|
| 2 |

**o. For each project, retrieve the project number, the project name, and the number of Employees who work on that project.**

```
SQL>  select Pnumber, Pname, count (*)
          from Project, Workson
          where Pnumber=Pno
              GROUP BY Pnumber, Pname;
```

**Explanation:** This query gives the pnumb ,pname and total number from project and workson tables using condition whose pnumber is pno and group by pnumber and pname.

**OUTPUT :**

| PNUMB | PNAME | COUNT(*) |
|---|---|---|
| p01 | Banking | 2 |
| p02 | Android App | 2 |
| p03 | WSN | 1 |
| p04 | Robotics | 1 |
| p05 | Smart Vehicle | 2 |

**4) Create an employee database with the fields: {eid, ename, dept, desig, salary, yoj, address {dno, street, locality, city}}**

Use employee

Db.createCollection(“emp”)

i. Insert 10 documents.

doc1 = {eid:002, ename:"kiran", dept:"production", desig:"hr", salary:40000, yoj:2016, address:{dno:47, street:3, locality:"JPnagar", city:"bangalore"} }

db.emp.insert(doc1)

ii. Display all the employees with salary in range (50000, 75000).

db.emp.find({$and:[{salary:{$gt:50000}},{salary:{$lt:75000}}]})

iii. Display all the employees with designation developer.

db.emp.find({desig:”developer”},{ename:1,desig:1,_id:0})

iv. Display the Salary of “Rahul”.

db.emp.find({ename:"Rahul"},{ename:1,salary:1,_id:0})

v. Display the city of employee “Rahul”.

db.emp.find({ename:"Rahul"},{ename:1,”address.city”:1,_id:0})

vi. Update the salary of developers by 5000 increment.

db.emp.update({desig:"developer"},{$inc:{"salary":5000}})

vii. Add field age to employee “Rahul”.

db.emp.update({ename:"Rahul"},{$set:{age:"22"}})

viii. Remove YOJ from “Rahul”.

db.emp.update({ename:"Rahul"},{$unset:{yoj:1}})

ix. Add an array field project to “Rahul”.

db.emp.update({ename:"Rahul"},{$push:{projects:"p1"}})

x. Add p2 and p3 project to “Rahul”.

db.emp.update({ename:"Rahul"},{$push:{projects:{$each:["p2","p3"]}}})

xi. Remove p3 from “Rahul”.

after puch its each for array insertion pf multiple values

db.emp.update({ename:"Rahul"},{$pull:{projects:"p3"}})

xii. Add a new embedded object “contacts” with “email” and “phone” as array objects to “Rahul”.

db.emp.update({ename:"Rahul"},{$push:{contacts:{phone:"9036240380",
email:"rahul@gmail.com"}}})

xiii. Add two phone numbers to “Rahul”.

db.emp.update({ename:"Rahul"},{$addToSet:{"contact.phone":[9738751143,988073 0784]}})

**5.Create a book Data Base with the fields: (isbn, bname, author [], year, publisher, price)**
**use Book**

db.createCollection(“book”)

Insert 5 documents.

doc1=({isbn:"e40", bname:"let us C", author:["yeshwanth", "kanaka"], year:2012, publisher:"pearson", price:100})

>db.book.insert(doc1)

List all the documents.

db.book.find().pretty()

List all book details except year and price.

>db.book.find({},{year:0,price:0})

Display all the books authored by rudresh.

>db.book.find({author:”rudresh”})

here c brackets

List all the books published by pearson.

>db.book.find({publisher:”pearson”})

List the publisher of book java.

>db.book.find({bname:”java”},{publisher:1,bname:1,_id:0})

List the author, publisher and year of the book “Let us C”.

>db.book.find({bname:”Let us C”},{bname:1, publisher:1,author:1,year:1})

Display the price of “Let us C” except _id.

>db.book.find({bname:”let us C”},{bname:1, price:1, _id:0})

Sort and display all books in ascending order of book names.

>db.book.find({},{bname:1}).sort({bname:1})

normal brackets and flower braces

Sort and display only 3 books in descending order of price.

>db.book.find().sort({price: -1}).limit(3)

Display all the books written by herbet and kuvempu.

db.book.find({author:{$all:[“herbert”,”kuvempu”]}},{bname:1,author:1,price:1})

there is author before all

Display all the books either written by herbet or kuvempu.

db.book.find({author:{$in:[“herbert”,”kuvempu”]}},{bname:1,author:1,price:1})

Display all the books where rama is the first author.

>db.book.find({"author.0":{$eq:"rama"}})

{id:0,bname:1}

**6. Create a Food Database with the fields: (food id, food cat, food name, chef name [ ], price, ingredients [], hotel name, hotel address {no, street, locality, city})**

Insert 10 documents.

doc1= {foodid:1, foodcat:"fastfood", foodname:"burger", chefname:["naveen","rakesh"], price:500, ingredients:["chees","corn"], hotelname:"mcburger", address:{no:31, street:"belroad", locality:"yelahanka", city:"bangalore"}}

db.food.insert(doc1)

List the price of pizza with ingredients.

db.food.find({foodname:"pizza"},{foodname:2, ingredients:1,price:1})

Display the item in the price range(500,800).

db.food.find({$and:[{price:{$lt:800}}, {price:{$gt:500}}]})

Display the item prepared by x and y.

db.food.find({chefname:{$all:["x","y"]}},{chefname:1,foodname:1, _id:0})

Display the item prepared by x or y.

db.food.find({chefname:{$in:["x", "y"]}},{ chefname:1,foodname:1, _id:0})

Add one chef to the food pizza.

db.food.update({foodname: "pizza"},{$push:{chef name: "rudresh"}})

Add ingredients to the food Burger.

db.food.update({foodname:"burger"},{$addToSet:{ingredients:["corn","cheese"]}})

Delete last ingredient added to the food burger.

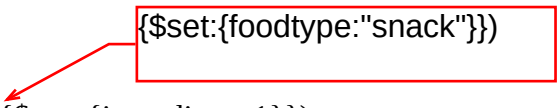
bcoz ingredients is array

db.food.update({foodname:"burger"},{$pop:{ingredients:1}})

Delete all the ingredients from the food biryani.

db.food.update({foodname:"biryani"},{$pullAll:{ingredients:[]}})

Add food type to the food Burger.




db.food.update({foodname:"burger"},{$pop:{ingredients:1}})

Modify the burger price by 200.

db.food.update({foodname:"burger"},{$set:{price:200}})

Add or insert a new food item with the food Id “f08 “ usingupsert as True.

db.food.update({foodid:"f08"},{foodid:"f08",foodcat:"chats",foodname:"pani

puri", chefname:["dhanush","pruthvi"],price:200,ingredients:["onion","puri"], hotelname:"chat street", address:{no:16,street:"8th",locality:"hebbal",city:"bangalore"}},true)

Increment the price of all food item in food cat: fastfood by 120.

db.food.update({foodcat:"fast food"},{$inc:{price:120}},false, true)

## 8. Lab Assignments

| SL. No. | Practice & Assignment Queries |
|---|---|
| 1 | **Assignment Queries for Lab Exercise I**<br><br>a. Retrieve the complete data from Customer table.<br>b. Retrieve the complete data from Agent table by mentioning attributes.<br>c. Display only product id and product name.<br>d. Display Product name and price as a single column and the column name be “Product and their prices” (concatenation operator and alias).<br>e. Display the city names of the customers by eliminating duplicates.<br>f. Retrieve the names of the customers lives in “Mumbai”.<br>g. Display agent ids and names belongs to New Delhi.<br>h. Retrieve customer ids who ordered both “P01” and “P02”.<br>i. Get customers whose name begins with letter "A".<br>j. Retrieve the customers whose name starts with letter “A” and third letter is “a” eg. Amar.<br>k. Retrieve the customers whose name consists of letter “a” eg. Amar, Anand.<br>l. Get customer ids whose discount is between 8 and 10.<br>m. Display the product name whose price is 10 or 20 using IN and OR operators.<br>n. Get total quantity of product "p01" that has been ordered.<br>o. Get number of cities in which customers are based.<br>p. Get total amount of all orders.<br>q. Get total number of customers.<br>r. Get average discount value for customers.<br>s. Get agent ids with the smallest percent commission.<br>t. Display the customer names who placed an order through the agent who is having aid as “a01”.<br>u. Retrieve the names of the customers who live in “Mumbai” and order product “P01”.<br>v. Retrieve customer ids who do not order part “P01”. |
| 3 | **Assignment Queries for Lab Exercise III**<br><br>a. List female employees from dept no is 2 earning more than Rs.35000<br>b. Retrieve the names and address of all employee who work for the ‘Research’ department.<br>c. Retrieve the names and salary of all employees who work in department number 5.<br>d. Retrieve the names of the employees and their superSSN name<br>e. Display name as "Employee name" and salary for the year as "Annual Income"<br>f. Display name, current salary and salary if it is going to be increased by 800 rupees<br>g. Display Department name and number as a single column with the name as “Department Details”<br>h. Retrieve the names of the managers who have more than two dependents.<br>i. Retrieve the names of the managers with atleast one dependent.<br>j. List all the Projects on which employee Ram is working<br>k. Retrieve the names of the employees who work on any project that Kumar works. |

| | |
|---|---|
| | l. Retrieve the names of the employees who do not have supervisor<br>m. Count the number of distinct salary values in the database.<br>n. For each project, retrieve the project number, the project name, and the number of employees from department 3 who work on the project.<br>o. Retrieve all employees in department 3 whose salary is between Rs.35,000 and Rs.40,000. |

## 9. Solutions for Lab Assignment Exercise I:

**a) Retrieve the complete data from customer table**

**SQL>** select * from customer;

**OUTPUT:**

| CID | CNAME | CITY | DISCOUNT |
|---|---|---|---|
| c001 | Sobhit | Darjeling | 10 |
| c002 | Bhanu | Srinagar | 12 |
| c003 | Amar | Srinagar | 8 |
| c004 | Anand | Darjeling | 8 |
| c005 | Anand | Mumbai | 0 |

**b) Retrieve the data complete from Agents by mentioning attributes**

**SQL>**select aid, aname, city, percent from agent;

**OUTPUT:**

| AID | ANAME | CITY | PERCENT |
|---|---|---|---|
| a01 | Sonu | NewDelhi | 6 |
| a02 | John | Agra | 6 |
| a03 | Bhargav | Jaipur | 7 |
| a04 | Gaurav | NewDelhi | 6 |
| a05 | Omkar | Srinagar | 5 |
| a06 | Sonu | Darjeling | 5 |

2 rows selected.

**c) Display only product id and product name**

**SQL>** select pid,pname from product;

**OUTPUT:**

| PID | PNAME |
|---|---|
| p01 | comb |
| p02 | brush |
| p03 | eraser |
| p04 | pen |
| p05 | pencil |
| p06 | folder |
| p07 | Highlighter |

7 rows selected.

**d) Display Product name and price as a single column and the column name be "Product and their prices" (concatenation operator and alias)**

**SQL>** select pname||price "product and their prices" from product;

**OUTPUT:**

| product and their prices | |
|---|---|
| comb | 10 |
| brush | 20 |
| eraser | 2 |
| pen | 15 |
| pencil | 3 |
| folder | 15 |
| Highlighter | 20 |

7 rows selected.

If some space has to be included between 2 column values after concatenating in the output, use number of spaces required within the single quote along with concatenation operator.

**SQL>** select pname||' '||price "product and their prices" from product;

**OUTPUT:**

| product and their prices | |
|---|---|
| comb | 10 |
| brush | 20 |
| eraser | 2 |
| pen | 15 |
| pencil | 3 |
| folder | 15 |
| Highlighter | 20 |

**SQL>** select pname||' cost is '||price "product and their prices" from product;

**OUTPUT:**

product and their prices

comb cost is 10
brush cost is 20
eraser cost is 2
pen cost is 15
pencil cost is 3
folder cost is 15
Highlighter cost is 20

**e) Display the city names of the customers by eliminating duplicates**

**SQL>** select distinct city from customer;

**OUTPUT:**

CITY

Srinagar
Darjeling
Mumbai
Srinagar

**f) Retrieve the names of the customers lives in "Mumbai"**

**SQL>** select cname from customer where city='Mumbai';

**OUTPUT:**

| CNAME |
|---|
| Anand |

**g) Display agent ids and names belongs to New Delhi**

**SQL>** select aid, aname from agent where city = 'NewDelhi';

**OUTPUT:**

| AID | ANAME |
|---|---|
| a01 | Sonu |
| a04 | Gaurav |

**h) Retrieve the customer ids who ordered the products "p01" and "p02"**

**SQL>** select cid from orders where pid='p01' intersect select cid from orders where pid='p07';

**OUTPUT:**

| CID |
|---|
| c001 |
| c005 |

**i) Get customers whose name begins with letter "A".**

**SQL>** select cname from customer where cname like 'A%';

**OUTPUT:**

| CNAME |
|---|
| Amar |
| Anand |
| Anand |

j)**Retrieve the customers whose name starts with letter "A" and third letter is "a" eg. Amar**

**SQL>** select cname from customer where cname like 'A_a%';

**OUTPUT:**

| CNAME |
|---|
| Amar |
| Anand |
| Anand |

**k)Retrieve the customers whose name consists of letter "a"**

**SQL>** select cname from customer where cname like '%a%';

**OUTPUT:**

| CNAME |
|---|
| Bhanu |
| Amar |
| Anand |

Anand

l)**Get customer ids whose discount is between 8 and 10.**

**SQL>** select cid from customer where discount between 8 and 10;

**OUTPUT:**

| CID |
|---|
| c001 |
| c003 |
| c004 |

**m) Display the product name whose price is 10 or 20 using IN and OR operators**

**SQL>** select pname from product where price in (10,20);

or

**SQL>** select pname from product where price=10 or price=20;

**OUTPUT:**

| PNAME |
|---|
| comb |
| brush |
| Highlighter |

**n) Get total quantity of product "p01" that has been ordered.**

**SQL>** select sum(qty) from orders where pid='p01'

**OUTPUT:**

| SUM(QTY) |
|---|
| 4800 |

**o) Get number of cities in which customers are based.**

**SQL>** select count ( distinct ( cname ) ) from customer;

**OUTPUT:**

| COUNT(DISTINCT(CNAME)) |
|---|
| 4 |

**p) Get total amount of all orders.**

**SQL>** select sum(ordamount) from orders;

**OUTPUT:**

| SUM(ORDAMOUNT) |
|---|
| 104598 |

**q)Get total number of customers.**

**SQL>** select count(cid) from customer;

**OUTPUT:**

| COUNT(CID) |
|---|
| 5 |

**r)Get average discount value for customers.**

**SQL>** select avg(discount) from customer;

**OUTPUT:**

| AVG(DISCOUNT) |
|---|
| 7.6 |

**s)Get agent ids with the smallest percent commission.**

**SQL>** select aid from agent where percent in (select min(percent) from agent);

Or select aid from agent where percent= (select min(percent) from agent)

**OUTPUT:**

| AID |
|---|
| a05 |
| a06 |

**t)Display the names of the customers who placed an order through the agent who is having aid as "a01"**

**SQL>** Select distinct(c.cname) from customer c , orders o where c.cid=o.cid and o.aid='a01';

**OUTPUT:**

| CNAME |
|---|
| Sobhit |
| Anand |

**u)Retrieve the names of the customers who live in "Mumbai" and order product "p01"**

**SQL>** select cname from customer c , orders o
where c.city='Mumbai' and c.cid=o.cid and o.pid='p01';

**OUTPUT:**

| CNAME |
|---|
| Anand |
| Anand |

**SQL>** select distinct(cname) from customer c,orders o
where c.city='Mumbai' and c.cid=o.cid and o.pid='p01';

**OUTPUT:**

| CNAME |
|---|
| Anand |

**v) Retrieve customer ids who do not order part "p01"**

**SQL>** select cid from customer minus (select cid from orders where pid='p01') ;
or
select cid from customer minus ( select distinct (cid) from orders where pid='p01');

**OUTPUT:**

| CID |
|---|
| c002 |

| c003 |
|---|

## 10. Viva Voce Questions

1) What is an RDBMS?
2) What is the SQL?
3) What are the different kinds of DBMS?
4) What are the features of relational database?
5) What are data types?
6) What is an E-R diagram?
7) What is the referential integrity?
8) What is a foreign key?
9) What is a primary key?
10) What is an alternate key in table?
11) What is the normalization?
12) Explain the First Normal Form?
13) Explain the Second Normal Form?
14) Explain the Third Normal Form?
15) What is an index, and how is it used to improve performance?
16) What are the types of indexes, and if separate indexes are created on each column of a table, what are the advantages and disadvantages of this approach?
17) What is the SQL Data Manipulation Language (DML)?
18) What is the SQL Data Definition Language (DDL)?
19) What is the de-normalization?
20) What is a transaction?
21) What are ACID properties?
22) What is the difference between DELETE TABLE and TRUNCATE TABLE commands?
23) What are constraints?
24) What are the different types of constraints?
25) What are cursors? What are the different types of cursors?
26) What are the advantages of cursors? How can you avoid cursors?
27) What is a join and explain different types of joins.
28) What is a self-join? Explain it with an example.
29) How do you implement one-to-one, one-to-many, many-to-many relationships while designing tables?
30) What is the difference between primary key and a unique key?
31) What are defaults?
32) What are triggers? How do you invoke a trigger on demand?
33) What is a stored procedure? What are the advantages?
34) What is the difference between stored procedure and a trigger?
35) What are the different types of parameters available in stored procedures?

36) How do you get the distinct rows in a table/ resultset?
37) How do you get the distinct rows without using the keyword DISTINCT?
38) How can you get the duplicated rows from the table using a single query?
39) How can you get the total number of records in a table?
40) How can you insert values in multiple rows using one insert statement?
41) What is the database replication?
42) What will happen when a Rollback statement is executed inside a Trigger?
43) What is collection in Mongo DB.
44) What is insert() method in mongo DB.
45) How to create a document in MongDB.

**Recommended Learning Resources:**


1. Raghu Ramakrishnan and Johannes Gehrke,*Database Management Systems*, 3rd Edition, McGraw-Hill, 2003.
2. Elmasri and Navathe,*Fundamentals of Database Systems*, 5th Edition, Pearson Education, 2007.
3. Shakuntala Gupta Edward, "Practical Mongo DB " Second edition, Apress Publications, 2016, ISBN 1484206487


**References:**


1. Abraham Silberschatz, Henry F. Korth, S. Sudarshan: "Database System Concepts", 6th Edition, McGraw Hill,

2010.

2. C J Date, "Database Design and Relational Theory: Normal Forms and All that Jazz", O 'Reilly, April 2012.

3. David Hows, "The definitive guide to MongoDB", 2nd edition, Apress Publication, 2009, 8132230485.


**JOURNALS/MAGAZINES:**

1. IEEE, IEEE Transactions on Knowledge and Data Engineering

2. Elsevier, Elsevier Data and Knowledge Engineering

3. ACM, ACM Transactions on Database Systems

**SWAYAM/NPTEL/MOOCs:**

1. https://www.coursera.org/courses?query=database

2. https://www.edx.org/learn/databases

3. https://academy.oracle.com/en/solutions-curriculum.html

**SELF-LEARNINGEXERCISES:**

1. Data warehousing, Data Marts, Getting data into the warehouse More exploration on GIThub

2. Data warehousing & KM , Data warehousing & CRM C modules interface