DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DATABASE MANAGEMENT SYSTEM

LAB MANUAL

V SEMESTER

ACADEMIC YEAR: 2023-2024

CS P53 DBMS LAB

1. Study of Database Concepts

Relational Model – Table – Operations On Tables –Index – Tablespace – Clusters – Synonym –View –Schema – Data Dictionary – Privilege – Role –Transactions

2. Study of SQL

Primitive Data Types – User Defined data Types – Built-in Functions –Parts of Speech of CREATE, ALTER, DROP, SELECT, INSERT, DELETE, UPDATE, COMMIT, ROLLBACK, SAVEPOINT, GRANT, and REVOKE

3. Study of Query Types

Queries involving Union, Intersection, Difference, Cartesian Product, Divide Operations – Sub Queries – Join Queries – Nested Queries –Correlated Queries – Recursive Queries

4. Application

Design and develop any two of the following
- Library Information System
- Logistics Management System
- Students' Information System
- Ticket Reservation System
- Hotel Management System
- Hospital Management System
- Inventory Control
- Retail Shop Management
- Employee Information System
- Payroll System
- Any other Similar System

Clearly mention the scope of the system. Use standard tools for expressing the design of the systems.

LIST OF EXPERIMENTS

EXPT. NO	EXPERIMENT NAME
1.	**DATABASE CONCEPTS**
1.1	Study of database concepts
2.	**STUDY OF SQL**
2.0	Implementation of BASIC commands
2.1	Implementation of DDL commands(without constraints)
2.2	Implementation of DDL commands(with constraints)
2.3	Implementation of DML commands (BASIC)
2.4	Implementation of DML commands (ADVANCED)
2.5	Implementation of DQL commands(BASIC)
2.6	Implementation of DQL commands(ADVANCED)
2.7	Implementation of DCL commands
2.8	Implementation of TCL commands
2.9	Built in functions
2.10	Implementation of AGGREGATE FUNCTIONS
3.	**STUDY OF SQL QUERIES**
3.1	SET OPERATIONS
3.2	JOINS
3.3	NESTED SUBQUERY
3.4	VIEW
3.5	TRIGGER
4.	**APPLICATION DEVELOPMENT**
4.1	**LIBRARY INFORMATION SYSTEM**
4.2	**LOGISTIC MANAGEMENT SYSTEM**
4.3	**EMPLOYEE DETAILS USING JDBC DATABASE CONNECTIVITY**

EX: NO: 1.1 STUDY OF DATABASE CONCEPTS

AIM

To make a study about the various database concepts and their structure.

DATABASE CONCEPTS

RELATIONAL MODEL – TABLE – OPERATIONS ON TABLES – INDEX – TABLESPACE – CLUSTERS – SYNONYM – VIEW – SCHEMA – DATA DICTIONARY – PRIVILEGE – ROLE – TRANSACTIONS

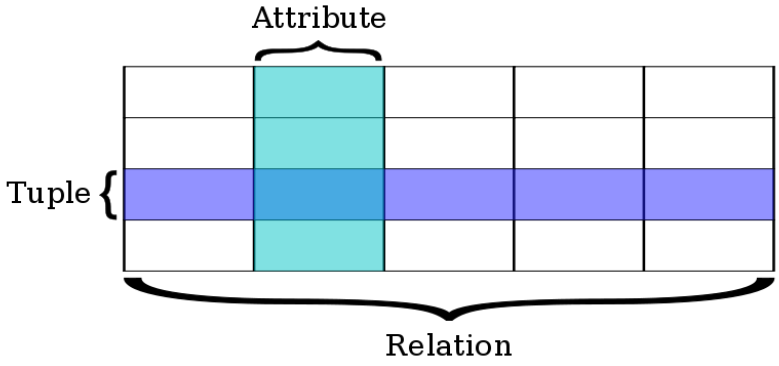
Relational Model

- Data and relationships are represented by a collection of **tables**.
- Each **table** has a number of columns with unique names, e.g. *customer*, *account*.

A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables:

- Values Are Atomic
- Each Row is Unique
- Column Values Are of the Same Kind
- The Sequence of Columns is Insignificant
- The Sequence of Rows is Insignificant
- Each Column Has a Unique Name



Example Instance of Student Relation

sid	name	login	age	Gpa
101	John	Jonh@cse	18	8.5
102	Sajan	sajan@mech	19	9.0
103	Rahul	rahul@it	18	9.5
104	smith	simth@it	18	9.2

1. STUDY OF SQL

Primitive Data Types – User Defined data Types – Built-in Functions – Parts of Speech of CREATE, ALTER, DROP, SELECT, INSERT, DELETE, UPDATE, COMMIT, ROLLBACK, SAVEPOINT, GRANT, REVOKE

SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL is an ANSI (American National Standards Institute) standard

SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

2. DATA TYPES

- Max. columns in a table is 255. Max.
- Char size is 255, Long is 64K & Number is 38 digits.
- Number(p,s) p is precision range 1 to 38, s is scale -84 to 127.
- Date Range from Jan 4712 BC to Dec 4712 AD.
- Raw Stores Binary data (Graphics Image & Digitized Sound). Max. is 255 bytes.

3. USER DEFINED TYPES

Two kinds of datatypes are internal and external.

- Internal datatypes specify storage of data in database columns.
- External datatypes specify how data is stored in host variables.

There are several kinds of user-defined types (UDTs). Some for each type and its constraints are shown below:

- Column types: These have no multiple column types but are valid only for single column. The code for a single column is shown below:
  - CREATE OR REPLACE TYPE NAME_T AS OBJECT (
    COL VARCHAR2 (30))

    - But the problem with column type is that the basic datatype is only valid for a column and if chained records are stored in more than one column then this type cannot be used.
- Multi-Column:
    - CREATE OR REPLACE TYPE ADDR_T AS OBJECT (
      ADDR1 VARCHAR2 (50),
      ADDR2 VARCHAR2 (50),
      CITY VARCHAR2 (30),
      STATE VARCHAR2 (2),
      ZIP_4 VARCHAR2(9));
    - Here each column can have different type but it is not applicable to individual fields.
- Row Types: These include single and multiple rows and form the foundation of object tables/views:
    - CREATE OR REPLACE TYPE EMP_T AS OBJECT (
      EMP_ID NUMBER (10),
      LNAME_TX NAME_T,
      FNAME_TX NAME_T,
      BIRTH_DATE DATE);
    - Not supported by query systems.
- Default values such as:
    - BOOLEAN_YN := 'Y'
      BOOLEAN_NY := 'N' are not supported.
- Constraints such as "AGE BETWEEN 18 AND 65" are hard to implement and check.

4. SQL – STRUCTURED QUERY LANGUAGE

Three Types

DDL
DML
DCL

4.1 DATA DEFINITION LANGUAGE (DDL)

- Specification notation for defining the database schema
- DDL compiler generates a set of tables stored in a *data dictionary*
- Data dictionary contains metadata (i.e., data about data)
    - Database schema
    - Data *storage and definition* language
        - Specifies the storage structure and access methods used
    - Integrity constraints
        - Domain constraints
        - Referential integrity (**references** constraint in SQL)
        - Assertions
    - Authorization

DDL Statements

- **Create table**
- **Alter table**
- **Drop table**

1. CREATE Statement

The CREATE TABLE statement is used to create a table

```
Create table table_name
(
        column_name1 data_type [constraints],
        column_name1 data_type [constraints],
        column_name1 data_type [constraints],
        ……..
);
```

2. Alter Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table

a. To Add a column

ALTER TABLE table_name ADD column_name datatype

b. To delete a column in a table

ALTER TABLE table_name DROP COLUMN column_name

c. To change the data type of a column in a table

ALTER TABLE table_name ALTER COLUMN column_name datatype

3. Drop Table

Used to delete the table permanently from the storage

DROP TABLE table_name

4.2 DATA MANIPULATION LANGUAGE (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as query language

- Two classes of languages
  - **Procedural** – user specifies what data is required and how to get those data

- **Declarative (nonprocedural)** – user specifies what data is required without specifying how to get those data

The query and update commands form the DML part of SQL:

- **SELECT** - extracts data from a table
- **UPDATE** - updates data in a table
- **DELETE** - deletes data from a table
- **INSERT INTO** - inserts new data into a table

The most important DDL statements in SQL are:

- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

An example of a table called "TeamIndia":

PlayerId	PlayerName	Address	TotCentury
1	Sachin	Mumbai	45
2	Dravid	Bangalore	25
3	Yuvaraj	Punjab	10

The table above contains three records (one for each person) and four columns

4.3 DATA CONTROL LANGUAGE

The **Data Control Language (DCL)** component of the SQL language is used to create privileges to allow users access to, and manipulation of, the database. There are two main commands:

GRANT to grant a privilege to a user
REVOKE to revoke (remove) a privilege from a user

GRANT command

In order to do anything within an Oracle database you must be given the appropriate privileges. Oracle operates a closed system in that you cannot perform any action at all unless you have been authorized to do so. This includes logging onto the database, creating tables, views, indexes and synonyms, manipulating data (ie select, insert, update and delete) in tables created by other users, etc.

The SQL command to grant a privilege on a table is:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON tablename TO username;
```

There are many more forms of the GRANT command, but this is sufficient for this Unit.

Any combination of the above privileges is allowed. You can issue this command on any tables that you have created. For example:

```
GRANT SELECT ON employee TO hn23;
GRANT SELECT, UPDATE, DELETE ON employee TO hn44;
```

REVOKE command

The SQL command to revoke a privilege on a table is:

REVOKE SELECT, INSERT, UPDATE, DELETE ON tablename FROM username;

For example:

```
REVOKE SELECT ON employee FROM hn23;
REVOKE SELECT, UPDATE, DELETE FROM hn44;
```

5. INDEX

A **database index** is a data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and increased storage space. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

5.1 Operations on Relations

Some important four mathematical set operations:

- The union operator combines the tuples of two relations and removes all duplicate tuples from the result. The relational union operator is equivalent to the SQL UNION operator.
- The intersection operator produces the set of tuples that two relations share in common. Intersection is implemented in SQL in the form of the INTERSECT operator.
- The difference operator acts on two relations and produces the set of tuples from the first relation that do not exist in the second relation. Difference is implemented in SQL in the form of the EXCEPT or MINUS operator.
- The cartesian product of two relations is a join that is not restricted by any criteria, resulting in every tuple of the first relation being matched with every tuple of the second relation. The cartesian product is implemented in SQL as the CROSS JOIN join operator.

6. TABLESPACE

A **tablespace** is a storage location where the actual data underlying database objects can be kept. It is the physical portion of the database used to allocate storage for all DBMS managed segments. A database segment is a database object which occupies physical space such as table data and indexes. Once created, a tablespace can be referred to by name when creating database segments.

7. VIEWS

A *view* is just a relation, but we store a *definition*, rather than a set of tuples.

```
CREATE VIEW YoungActiveStudents (name, grade)
AS SELECT   S.name, E.grade
FROM Students S, Enrolled E
WHERE S.sid = E.sid and S.age<21
```

Views can be dropped using the DROP VIEW command.

How to handle DROP TABLE if there's a view on the table?

DROP TABLE command has options to let the user specify this.

7.1 VIEWS AND SECURITY

Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).

Given YoungStudents, but not Students or Enrolled, we can find students s who have are enrolled, but not the *cid's* of the courses they are enrolled in.

8. ROLE AND PRIVILEGE

A **role** is a set or group of privileges that can be granted to users or another role.

Creating a Role

To create a role, we must have CREATE ROLE system privileges. The syntax for creating a role is:

CREATEROLE role_name

[NOT IDENTIFIED | IDENTIFIED {BY password | USING [schema.] package |

EXTERNALLY | GLOBALLY } ;

- The **role_name** phrase is the name of the new role that we are creating. This is how we will refer to the grouping of privileges.
- The **NOT IDENTIFIED** phrase means that the role is immediately enabled. No password is required to enable the role.
- The **IDENTIFIED** phrase means that a user must be authorized by a specified method before the role is enabled.
- The **BY password** phrase means that a user must supply a password to enable the role.

- The **USING package** phrase means that you are creating an application role - a role that is enabled only by applications using an authorized package.
- The **EXTERNALLY** phrase means that a user must be authorized by an external service to enable the role. An external service can be an operating system or third-party service.
- The **GLOBALLY** phrase means that a user must be authorized by the enterprise directory service to enable the role.

For example: **CREATE ROLE test_role;**
This first example creates a role called test_role.

CREATE ROLE test_role

IDENTIFIED BY test123;

This second example creates the same role called test_role, but now it is password protected with the password of test123.

8.1 Grant Privileges (on Tables) to Roles

We can grant roles various privileges to tables. These privileges can be any combination of select, insert, update, delete, references, alter, and index. Below is an explanation of what each privilege means.

Privilege	Description
Select	Ability to query the table with a select statement.
Insert	Ability to add new rows to the table with the insert statement.
Update	Ability to update rows in the table with the update statement.
Delete	Ability to delete rows from the table with the delete statement.
References	Ability to create a constraint that refers to the table.
Alter	Ability to change the table definition with the alter table statement.
Index	Ability to create an index on the table with the create index statement.

The syntax for granting privileges on a table is:

grant privileges on object to role_name

For example, if we wanted to grant select, insert, update, and delete privileges on a table called suppliers to a role named test_role, we would execute the following statement:

grant select, insert, update, delete on suppliers to test_role;

We can also use the all keyword to indicate that you wish all permissions to be granted. For example: **grant all on suppliers to test_role;**

Revoke Privileges (on Tables) to Roles: Once we have granted privileges, we may need to revoke some or all of these privileges. To do this, we can execute a revoke command. We can revoke any combination of select, insert, update, delete, references, alter, and index. The syntax for revoking privileges on a table is:

revoke privileges on object from role_name;

For example, if we wanted to revoke delete privileges on a table called suppliers from a role named test_role, we would execute the following statement:

revoke delete on suppliers from test_role;

If we wanted to revoke all privileges on a table, we could use the all keyword. For example:

revoke all on suppliers from test_role;

8.2 Grant Privileges (on Functions/Procedures) to Roles

When dealing with functions and procedures, we can grant roles the ability to execute these functions and procedures. The Execute privilege is explained below:

Privilege	Description
Execute	Ability to compile the function/procedure. Ability to execute the function/procedure directly.

The syntax for granting execute privileges on a function/procedure is:

grant execute on object to role_name;

For example, if we had a function called Find_Value and we wanted to grant execute access to the role named test_role, we would execute the following statement:

grant execute on Find_Value to test_role;

8.3 Revoke Privileges (on Functions/Procedures) to Roles

Once we have granted execute privileges on a function or procedure, we may need to revoke these privileges from a role. To do this, we can execute a revoke command. The syntax for the revoking privileges on a function or procedure is:

revoke execute on object from role_name;

If we wanted to revoke execute privileges on a function called Find_Value from a role named test_role, we would execute the following statement:

revoke execute on Find_Value from test_role;

8.4 Granting the Role to a User

Now, that we have created the role and assigned the privileges to the role, we will need to grant the role to specific users. The syntax to grant a role to a user is:

GRANT role_name TO user_name;

For example: **GRANT test_role to smithj;**
This example would grant the role called test_role to the user named smithj.

The SET ROLE statement

The SET ROLE statement allows you to enable or disable a role for a current session.
When a user logs into Oracle, all *default* roles are enabled, but non-default roles must be enabled with the SET ROLE statement.
The syntax for the SET ROLE statement is:

SET ROLE (role_name [IDENTIFIED BY password] | ALL [EXCEPT role1, role2, ...] | NONE);

- The **role_name** phrase is the name of the role that you wish to enable.
- The **IDENTIFIED BY password** phrase is the password for the role to enable it. If the role does not have a password, this phrase can be omitted.
- The **ALL** phrase means that all roles should be enabled for this current session, except those listed in the **EXCEPT** phrase.
- The **NONE** phrase disables all roles for the current session. (including all default roles)

For example: **SET ROLE test_role IDENTIFIED BY test123;**
This example would enable the role called test_role with a password of test123.

8.5 Setting a role as DEFAULT Role

A default role means that the role is always enabled for the current session at logon. It is not necessary to issue the SET ROLE statement. To set a role as a DEFAULT role, we need to issue the ALTER USER statement. The syntax for setting a role as a DEFAULT role is:

ALTER USER user_name DEFAULT ROLE (role_name | ALL [EXCEPT role1, role2, ...] | NONE);

- The **user_name** phrase is the name of the user whose role we are setting as DEFAULT.
- The **role_name** phrase is the name of the role that we wish to set as DEFAULT.
- The **ALL** phrase means that all roles should be enabled as DEFAULT, except those listed in the **EXCEPT** phrase.

The **NONE** phrase disables all roles as DEFAULT. For example:

ALTER USER smithj DEFAULT ROLE test_role;

This example would set the role called test_role as a DEFAULT role for the user named smithj.

ALTER USER smithj DEFAULT ROLE ALL;

This example would set all roles assigned to smithj as DEFAULT.

ALTER USER smithj DEFAULT ROLE ALL EXCEPT test_role;

This example would set all roles assigned to smithj as DEFAULT, except for the role called test_role.

8.6 Dropping a Role

It is also possible to drop a role. The syntax for dropping a role is:

DROP ROLE role_name;

For example: **DROP ROLE test_role;**

This drop statement would drop the role called *test_role* that we defined earlier.

9. TRANSACTIONS

A **transaction** is a logical unit of work that contains one or more SQL statements. The effects of all the SQL statements in a transaction can be either all **committed** (applied to the database) or all **rolled back** (undone from the database).

A transaction begins with the first executable SQL statement. A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL statement is issued.

A transaction ends when any of the following occurs:

- A user issues a COMMIT or ROLLBACK statement without a SAVEPOINT clause.
- A user runs a DDL statement such as CREATE, DROP, RENAME, or ALTER. If the current transaction contains any DML statements, Oracle first commits the transaction, and then runs and commits the DDL statement as a new, single statement transaction.
- A user disconnects from Oracle. The current transaction is committed.
- A user process terminates abnormally. The current transaction is rolled back.

After one transaction ends, the next executable SQL statement automatically starts the following transaction.

9.1 COMMIT TRANSACTIONS

Committing a transaction means making permanent the changes performed by the SQL statements within the transaction.

Before a transaction that modifies data is committed, the following has occurred:

- Oracle has generated rollback segment records in buffers in the SGA that store rollback segment data. The rollback information contains the old data values changed by the SQL statements of the transaction.
- Oracle has generated redo log entries in the redo log buffer of the SGA. The redo log record contains the change to the data block and the change to the rollback block. These changes may go to disk before a transaction is committed.
- The changes have been made to the database buffers of the SGA. These changes may go to disk before a transaction is committed.

When a transaction is committed, the following occurs:

1. The internal transaction table for the associated rollback segment records that the transaction has committed, and the corresponding unique system change number (SCN) of the transaction is assigned and recorded in the table.
2. The log writer process (LGWR) writes redo log entries in the SGA's redo log buffers to the online redo log file. It also writes the transaction's SCN to the online redo log file. This atomic event constitutes the commit of the transaction.
3. Oracle releases locks held on rows and tables.
4. Oracle marks the transaction complete.

9.2 ROLLBACK OF TRANSACTIONS

Rolling back means undoing any changes to data that have been performed by SQL statements within an uncommitted transaction. Oracle uses undo tablespaces or rollback segments to store old values. We can also roll back the trailing portion of an uncommitted transaction to a marker called a savepoint.

All types of rollbacks use the same procedures:

- Statement-level rollback (due to statement or deadlock execution error)
- Rollback to a savepoint
- Rollback of a transaction due to user request
- Rollback of a transaction due to abnormal process termination
- Rollback of all outstanding transactions when an instance terminates abnormally
- Rollback of incomplete transactions during recovery

In rolling back **an entire transaction**, without referencing any savepoints, the following occurs:

1. Oracle undoes all changes made by all the SQL statements in the transaction by using the corresponding undo tablespace or rollback segment.
2. Oracle releases all the transaction's locks of data.
3. The transaction ends.

9.3 SAVEPOINTS IN TRANSACTIONS

We can declare intermediate markers called **savepoints** within the context of a transaction. Savepoints divide a long transaction into smaller parts.

Using savepoints, we can arbitrarily mark our work at any point within a long transaction. We then have the option later of rolling back work performed before the current point in the transaction but after a declared savepoint within the transaction. After a rollback to a savepoint, Oracle releases the data locks obtained by rolled back statements. Other transactions that were waiting for the previously locked resources can proceed. Other transactions that want to update previously locked rows can do so.

When a transaction is rolled back to a savepoint, the following occurs:

1. Oracle rolls back only the statements run after the savepoint.
2. Oracle preserves the specified savepoint, but all savepoints that were established after the specified one are lost.
3. Oracle releases all table and row locks acquired since that savepoint but retains all data locks acquired previous to the savepoint.

The transaction remains active and can be continued.

RESULT

Thus the various database concepts have been studied.

2. STUDY OF SQL

EX: NO: 2.0 IMPLEMENTATION OF BASIC COMMANDS

AIM:

To create a table named "Emp" with necessary columns and then perform the Basic commands.

ALGORITHM:

STEP 1: Start.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop.

CREATING THE TABLE:

```
SQL> CREATE TABLE  emp
        (
                empno NUMBER,
                empname VARCHAR2(15),
                DOB DATE,
                salary NUMBER,
                designation VARCHAR2(10)
        );

        Table created.
```

INSERTING THE VALUES IN THE TABLE:

//insert values in the table emp:

```
SQL> INSERT INTO emp (empno, empname, DOB, salary, designation) VALUES(101,'Greg',
'20-JUL-1994',25000,'Clerk');
1 row created.

SQL> INSERT INTO emp(empno,empname, DOB, salary, designation) VALUES(100,'John',
'21-APR-1996',50000,'Manager');
1 row created.
```

DESCRIBING THE TABLE:

//displaying the values in the table:

SQL> select * from emp;

EMPNO	EMPNAME	DOB	SALARY	DESIGNATION
101	Greg	20-JUL-94	25000	Clerk
100	John	21-APR-96	50000	Manager

//To display the empname and salary from the table emp

SQL> SELECT empname, salary FROM emp;

EMPNAME	SALARY
Greg	25000
John	50000

//To display the empname and salary where salary is greater than 5000

SQL> SELECT empname,salary FROM emp WHERE salary > 5000;

EMPNAME	SALARY
Greg	25000
John	50000

MODIFY VALUES:

SQL> UPDATE emp SET salary = salary + 1000;
2 rows updated.

SQL> select * from emp;

EMPNO	EMPNAME	DOB	SALARY	DESIGNATION
101	Greg	20-JUL-94	26000	Clerk
100	John	21-APR-96	51000	Manager

DELETE VALUES:

SQL> DELETE FROM emp WHERE empno=100;
1 row deleted.

SQL> select * from emp;

EMPNO	EMPNAME	DOB	SALARY	DESIGNATIO
101	Greg	20-JUL-94	26000	Clerk

Result:

Thus, the various SQL Basic commands was studied and executed successfully.

EX: NO: 2.1 IMPLEMENTATION OF DDL COMMANDS

AIM:

To create a table named "Emp" with necessary columns and then perform the DDL commands without constraint.

ALGORITHM:

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

DESCRIPTION AND SYNTAX

1. **CREATE COMMAND:-**

Syntax:

Create table <table_name> (column_name datatype (size) constrains);

Description

The create command when applied with above specification creates the field of different data type.

2. **ALTER COMMAND:-**

Syntax:

a) alter table <table_name> add (column_name datatype (size));

Description

The alter command when used with add allows us to add an additional column to an already existing table.

Syntax:

b) alter table < table_name> modify(column_name datatype(size));

Description

The alter command when used with modify redefines the column with the given values but cannot change the column names.

Syntax:

c) alter table <table_name> drop(column_name);

Description

The alter command when used with drop deletes the specified column in the table.

3. DROP COMMAND:-

Syntax:

Drop table <table_name>;

Description

A table can be dropped (deleted) by using a drop table command.

4. CREATE VIEW COMMAND:-

Syntax:

Create view <view_name> as select <column_name> from <table_name> where <condition>;

Description

A view is named, derived, virtual table. A view takes the output of a query and treats it as a table; a view can be thought of as a "stored query" or a "virtual table". The tables upon which a view is based are called base tables.

5. DROP VIEW COMMAND:-

Syntax:

Drop view <view_name>;

Description

A view can be dropped (deleted) by using a drop view command.

6. TRUNCATE COMMAND:-

Syntax:
Truncate table <table_name>;

Description

The details in the table are deleted but the table structure remains.

7. RENAME COMMAND:-

Syntax:
Rename <oldtable_name> to <newtable_nmae>;

Description

The old table name is replaced with the new table name.

PROGRAM TO LEARN DDL COMMANDS

CREATE A TABLE:

```
SQL> CREATE TABLE emp
      (
            empno NUMBER,
            empname VARCHAR2(15),
            dob DATE,
            salary NUMBER,
            designation VARCHAR2(15)
      );
```

Table created.

DESCRIBE THE TABLE EMP:

SQL> DESC emp;

Name	Null?	Type
EMPNO		NUMBER
EMPNAME		VARCHAR2(15)
DOB		DATE
SALARY		NUMBER

DESIGNATION VARCHAR2(15)

ALTER THE TABLE

A. ADD

//To alter the table emp by adding new attribute department

SQL> ALTER TABLE emp ADD department VARCHAR2(15);
Table altered.

SQL> DESC emp;

Name	Null?	Type
EMPNO		NUMBER
EMPNAME		VARCHAR2(15)
DOB		DATE
SALARY		NUMBER
DESIGNATION		VARCHAR2(15)
DEPARTMENT		VARCHAR2(15)

B. MODIFY

// To alter the table emp by modifying the size of the attribute department

SQL> ALTER TABLE emp MODIFY department VARCHAR2(30);
Table altered.

SQL> DESC emp;

Name	Null?	Type
EMPNO		NUMBER
EMPNAME		VARCHAR2(15)
DOB		DATE
SALARY		NUMBER
DESIGNATION		VARCHAR2(15)
DEPARTMENT		VARCHAR2(30)

C. DROP

// To alter the table emp by deleting the attribute department

SQL> ALTER TABLE emp DROP (department);
Table altered.

SQL> DESC emp;

Name	Null?	Type
EMPNO		NUMBER
EMPNAME		VARCHAR2(15)
DOB		DATE
SALARY		NUMBER
DESIGNATION		VARCHAR2(15)

D. RENAME

// To alter the table name by using rename keyword

SQL> ALTER TABLE emp RENAME TO emp1 ;
Table altered.

SQL> DESC emp1;

Name	Null?	Type
EMPNO		NUMBER
EMPNAME		VARCHAR2(15)
DOB		DATE
SALARY		NUMBER
DESIGNATION		VARCHAR2(15)

DROP TABLE:

//To delete the table from the database
SQL> DROP TABLE emp1;
Table dropped.

SQL> DESC emp1;
ERROR:
ORA-04043: object emp1 does not exist

Result:

Thus, the various commands in Data definition language without constrains was studied and executed successfully.

EX: NO: 2.2 IMPLEMENTATION OF DDL WITH CONSTRAINT

AIM

To create a table named "Student and Exam" with necessary columns and then perform the DDL commands with constraint.

ALGORITHM:

STEP 1: Start.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop.

DDL WITH CONSTRAINTS:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

PROGRAM TO LEARN DDL COMMAND WITH CONSTRAINT

CREATE THE TABLE
// To create a table student

```
SQL> CREATE TABLE student
        (
                studentID NUMBER PRIMARY KEY,
                sname VARCHAR2(15) NOT NULL,
                department CHAR(5),
                sem NUMBER,
                dob DATE,
                email_id VARCHAR2(20) UNIQUE,
                college VARCHAR2(10) DEFAULT 'MVIT'
        );

        Table created.
```

// Describe the table student

```
SQL> DESC student;
```

Name	Null?	Type
STUDENTID	NOT NULL	NUMBER
SNAME	NOT NULL	VARCHAR2(15)
DEPARTMENT		CHAR(5)
SEM		NUMBER
DOB		DATE
EMAIL_ID		VARCHAR2(20)
COLLEGE		VARCHAR2(10)

//To create a table exam

```
SQL> CREATE TABLE exam
        (
                examID NUMBER PRIMARY KEY,
                studentID NUMBER REFERENCES student(studentID),
                department CHAR(5) NOT NULL,
                mark1 NUMBER CHECK (mark1<=100 and mark1>=0),
                mark2 NUMBER CHECK (mark2<=100 and mark2>=0),
                mark3 NUMBER CHECK (mark3<=100 and mark3>=0),
                mark4 NUMBER CHECK (mark4<=100 and mark4>=0),
                mark5 NUMBER CHECK (mark5<=100 and mark5>=0),
                total NUMBER,
                average NUMBER,
                grade CHAR(1)
        );
        Table created.
```

// Describe the table student

```
SQL> DESC exam;
```

Name	Null?	Type
EXAMID	NOT NULL	NUMBER
STUDENTID		NUMBER
DEPARTMENT	NOT NULL	CHAR(5)
MARK1		NUMBER
MARK2		NUMBER
MARK3		NUMBER
MARK4		NUMBER
MARK5		NUMBER
TOTAL		NUMBER
AVERAGE		NUMBER
GRADE		CHAR(1)

ALTER THE TABLE:

A. ADD
// To alter the table student by adding new attribute address

SQL> ALTER TABLE student ADD address VARCHAR2(40);
Table altered.

SQL> DESC student;

Name	Null?	Type
STUDENTID	NOT NULL	NUMBER
SNAME	NOT NULL	VARCHAR2(15)
DEPARTMENT		CHAR(5)
SEM		NUMBER
DOB		DATE
EMAIL_ID		VARCHAR2(20)
COLLEGE		VARCHAR2(10)
ADDRESS		VARCHAR2(40)

B. MODIFY
// To alter the table student by modifying the size of the attribute address

SQL> ALTER TABLE student MODIFY address VARCHAR2(80);
Table altered.

SQL> DESC student;

Name	Null?	Type
STUDENTID	NOT NULL	NUMBER
SNAME	NOT NULL	VARCHAR2(15)
DEPARTMENT		CHAR(5)
SEM		NUMBER
DOB		DATE
EMAIL_ID		VARCHAR2(20)
COLLEGE		VARCHAR2(10)
ADDRESS		VARCHAR2(80)

C. DROP
// To alter the table student by deleting the attribute address

SQL> ALTER TABLE student DROP(address);
Table altered.

SQL> DESC student;

Name	Null?	Type
STUDENTID	NOT NULL	NUMBER
SNAME	NOT NULL	VARCHAR2(15)
DEPARTMENT		CHAR(5)
SEM		NUMBER
DOB		DATE
EMAIL_ID		VARCHAR2(20)
COLLEGE		VARCHAR2(10)

D. RENAME
// To alter the table name by using rename keyword

SQL> ALTER TABLE student RENAME TO student1 ;
Table altered.

SQL> DESC student1;

Name	Null?	Type
STUDENTID	NOT NULL	NUMBER
SNAME	NOT NULL	VARCHAR2(15)
DEPARTMENT		CHAR(5)
SEM		NUMBER
DOB		DATE
EMAIL_ID		VARCHAR2(20)
COLLEGE		VARCHAR2(10)

DROP TABLE:
// To delete the table from the database

SQL> DROP TABLE exam;
Table dropped.

SQL> DESC exam;
ERROR:
ORA-04043: object exam does not exist

Result:

Thus, the various commands in Data definition language with constrains was studied and executed successfully.

EX: NO: 2.3 IMPLEMENTATION OF DML BASIC COMMANDS

AIM

To create a table named "Ship" with necessary columns and then perform the DML Basic commands.

PROCEDURE

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

The DML commands are:

- Insert
- Delete
- Update
- Select

INSERT:-

Syntax :

Insert into <table_name> values (val1,val2,…);

Description:

The 'insert into' command insert the values in the specified table. In the insert into SQL sentence the column and values have a one to one relationship (i.e.) the first value described into the first column, the second value described being inserted into the second column and so on.

DELETE:-

Syntax:

Delete from <table_name> where <condition>;

Description:

The delete in SQL is used to remove rows from table.

To remove,

1. All the rows from a table.
2. A select set of rows from a table.

UPDATE:-

Syntax:
Update <table_name> set fieldname=<expression> where <condition>;

Description:

The update command is used to change or modify data values in a table.
To update,

1. All the rows from a table.
2. A select set of rows from a table.

SELECT:-

Syntax:
a) select <attribute lists> from <table_name> [where clause];

Description:

Select command is used to retrieve data from one or more tables or columns. The attributes list is a list of attributes name whose values are displayed by query. A missing where clauses indicate no condition on tuples selection. The condition is a Boolean expression that identifies the tuples to be retrieved by the query.

Syntax:
b) select distinct <column_name> from <table_name>;

Description:

Display the distinct values from a table by eliminating the duplicate values. It performs grouping of the specified fields when queried with distinct statement.

PROGRAM TO LEARN DML COMMANDS

CREATE TABLE

```
SQL> create table ship
    (
        shipid NUMBER,
        shipname VARCHAR2(10),
        origin VARCHAR2(15),
        destination VARCHAR2(15),
        passenger NUMBER
    );
Table created.
```

INSERT THE VALUES IN TABLE SHIP

```
SQL> INSERT INTO ship VALUES(101, 'titanic', 'america', 'canada', 528);
1 row created.

SQL> INSERT INTO ship VALUES(102, 'alligator', 'america', 'africa', 524);
1 row created.

SQL> INSERT INTO ship VALUES(103, 'cairo', 'german', 'india', 628);
1 row created.

SQL> INSERT INTO ship VALUES(104, 'costa', 'italy', 'german', 438);
1 row created.

SQL> INSERT INTO ship VALUES(105, 'daigo', 'japan', 'india', 724);
1 row created.

SQL> INSERT INTO ship VALUES(106, 'eclipse', 'german', 'italy', 510);
1 row created.

SQL> INSERT INTO ship VALUES(107, 'blue', 'canada', 'danish', 865);
1 row created.
```

DESCRIBING THE TABLE

```
SQL> SELECT * FROM ship;
```

SHIPID	SHIPNAME	ORIGIN	DESTINATION	PASSENGER
101	titanic	america	canada	528
102	alligator	america	africa	524
103	cairo	german	india	628
104	costa	italy	german	438
105	daigo	japan	india	724
106	eclipse	german	italy	510
107	blue	canada	danish	865

7 rows selected.

UPDATE COMMAND

```
SQL> UPDATE ship SET origin='russia' WHERE shipid=102;
1 row updated.
```

SQL> select * from ship;

SHIPID	SHIPNAME	ORIGIN	DESTINATION	PASSENGER
101	titanic	america	canada	528
102	alligator	russia	africa	524
103	cairo	german	india	628
104	costa	italy	german	438
105	daigo	japan	india	724
106	eclipse	german	italy	510
107	blue	canada	danish	865

7 rows selected.

DELETE COMMAND

SQL> DELETE FROM ship WHERE shipid=107;
1 row deleted.

SQL> select * from ship;

SHIPID	SHIPNAME	ORIGIN	DESTINATION	PASSENGER
101	titanic	america	canada	528
102	alligator	russia	africa	524
103	cairo	german	india	628
104	costa	italy	german	438
105	daigo	japan	india	724
106	eclipse	german	italy	510

6 rows selected.

Result:

Thus, the various Basic commands in Data Manipulation language was studied and executed successfully.

EX: NO: 2.4 IMPLEMENTATION OF DML COMMANDS (with constraint)

AIM

To create a table named "Student and Exam" with necessary columns and then perform the DML commands with Constraint.

PROCEDURE

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

PROGRAM TO LEARN DML WITH CONSTRAINT

CREATE THE TABLE

// To create a table student

```
SQL> CREATE TABLE student
        (
                SID NUMBER PRIMARY KEY,
                sname VARCHAR2(15) NOT NULL,
                dept CHAR(5),
                sem NUMBER,
                dob DATE,
                email_id VARCHAR2(20) UNIQUE,
                college VARCHAR2(10) DEFAULT 'MVIT'
        );
Table created.
```

// To create a table exam

```
SQL> CREATE TABLE exam
        (
                EID NUMBER PRIMARY KEY,
                SID NUMBER REFERENCES student(SID),
                dept CHAR(5) NOT NULL,
                m1 NUMBER CHECK (m1<=100 and m1>=0),
                m2 NUMBER CHECK (m2<=100 and m2>=0),
                m3 NUMBER CHECK (m3<=100 and m3>=0),
                m4 NUMBER CHECK (m4<=100 and m4>=0),
                m5 NUMBER CHECK (m5<=100 and m5>=0),
                tot NUMBER,
                avg NUMBER,
                grade CHAR(1)
        );
```

Table created.

INSERTING VALUES:

//insert values into student table:

SQL>INSERT INTO student VALUES (101,'RUPESH','IT', 5,'5-JUN-2005',
'rupesh@gmail.com','MEC');
1 row created.

SQL> INSERT INTO student VALUES (102,'BALA','CSE',7,'7-OCT-1995','bala@gmail.com','IIT');
1 row created.

SQL> INSERT INTO student VALUES (104,'HEMESH','IT',5,'23-JUL-1996','hemesh@gmail.com','IIT');
1 row created.

SQL> INSERT INTO student VALUES (106,'SAIVAISHNAVI','CSE',5,'9-JUN-1996',
'vaishu@gmail.com','SMVEC');
1 row created.

SQL> INSERT INTO student VALUES (108,'RISHA','IT',5,'21-APR-1996','risha@gmail.com','');
1 row created.

SQL> SELECT * FROM student;

SID	SNAME	DEPT	SEM	DOB	EMAIL_ID	COLLEGE
101	RUPESH	IT	5	05-JUN-05	rupesh@gmail.com	MEC
102	BALA	CSE	7	07-OCT-95	bala@gmail.com	IIT
104	HEMESH	IT	5	23-JUL-96	hemesh@gmail.com	IIT
106	SAIVAISHNAVI	CSE	5	09-JUN-96	vaishu@gmail.com	SMVEC
108	RISHA	IT	5	21-APR-96	risha@gmail.com	MVIT

//insert value into exam table:

SQL> INSERT INTO exam(eid, sid, dept, m1, m2, m3, m4, m5)VALUES (2222,101,'IT',98,87,83,99,87);
1 row created.

SQL> INSERT INTO exam(eid, sid, dept, m1, m2, m3, m4,m5)VALUES(3333,104,'IT',99,82,84,89,100);
1 row created.

SQL> INSERT INTO exam(eid, sid, dept, m1, m2, m3, m4,m5)VALUES (4444,108,'IT',92,85,83,91,87);
1 row created.

SQL> INSERT INTO exam(eid, sid, dept, m1, m2, m3, m4,m5)VALUES(5555,106,'CSE',82,85,87,91,85);
1 row created.

SQL> SELECT * FROM exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
2222	101	IT	98	87	83	99	87			
3333	104	IT	99	82	84	89	100			
4444	108	IT	92	85	83	91	87			
5555	106	CSE	82	85	87	91	85			

2. UPDATE:

// To change the values in the table student

SQL> UPDATE student SET college='MEC' WHERE sid=108;
1 row updated.

SQL> SELECT * FROM student;

SID	SNAME	DEPT	SEM	DOB	EMAIL_ID	COLLEGE
101	RUPESH	IT	5	05-JUN-05	rupesh@gmail.com	MEC
102	BALA	CSE	7	07-OCT-95	bala@gmail.com	IIT
104	HEMESH	IT	5	23-JUL-96	hemesh@gmail.com	IIT
106	SAIVAISHNAVI	CSE	5	09-JUN-96	vaishu@gmail.com	SMVEC
108	RISHA	IT	5	21-APR-96	risha@gmail.com	MEC

// To set the total in the table exam

SQL> UPDATE exam SET tot=(m1+m2+m3+m4+m5);
4 rows updated.

SQL> select * from exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
2222	101	IT	98	87	83	99	87	454		
3333	104	IT	99	82	84	89	100	454		
4444	108	IT	92	85	83	91	87	438		
5555	106	CSE	82	85	87	91	85	430		

// To set the average in the table exam

SQL> UPDATE exam SET avg=tot/5;
4 rows updated.

SQL> select * from exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE

2222	101	IT	98	87	83	99	87	454	90.8
3333	104	IT	99	82	84	89	100	454	90.8
4444	108	IT	92	85	83	91	87	438	87.6
5555	106	CSE	82	85	87	91	85	430	86

// To set the grade in the table exam

```
SQL> UPDATE exam SET grade='S' WHERE avg>95;
0 rows updated.

SQL> UPDATE exam SET grade='A' WHERE avg<=95 AND avg>90;
2 rows updated.

SQL> UPDATE exam SET grade='B' WHERE avg<=90 AND avg>85;
2 rows updated.

SQL> UPDATE exam SET grade='C' WHERE avg<=85 AND avg>80;
0 rows updated.

SQL> UPDATE exam SET grade='D' WHERE avg<=80 AND avg>75;
0 rows updated.

SQL> UPDATE exam SET grade='F' WHERE avg<75;
0 rows updated.

SQL> select * from exam;
```

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
2222	101	IT	98	87	83	99	87	454	90.8	A
3333	104	IT	99	82	84	89	100	454	90.8	A
4444	108	IT	92	85	83	91	87	438	87.6	B
5555	106	CSE	82	85	87	91	85	430	86	B

// To delete a particular record whose the exam id is 2222

```
SQL> DELETE FROM exam WHERE eid=2222;
1 row deleted.
 SQL> select * from exam;
```

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
3333	104	IT	99	82	84	89	100	454	90.8	A
4444	108	IT	92	85	83	91	87	438	87.6	B
5555	106	CSE	82	85	87	91	85	430	86	B

// To inserted the same record in table exam for further use

SQL>INSERT INTO exam(eid, sid, dept, m1, m2, m3, m4, m5) VALUES (2222,101,'IT', 98,87,83,99,87);
1 row created.

SQL> SELECT * FROM exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
3333	104	IT	99	82	84	89	100	454	90.8	A
4444	108	IT	92	85	83	91	87	438	87.6	B
5555	106	CSE	82	85	87	91	85	430	86	B
2222	101	IT	98	87	83	99	87			

SQL> UPDATE exam SET tot=(m1+m2+m3+m4+m5) where EID=2222;
1 row updated.

SQL> UPDATE exam SET avg=tot/5 where EID=2222;
1 row updated.

SQL> select * from exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
3333	104	IT	99	82	84	89	100	454	90.8	A
4444	108	IT	92	85	83	91	87	438	87.6	B
5555	106	CSE	82	85	87	91	85	430	86	B
2222	101	IT	98	87	83	99	87	454	90.8	

SQL> UPDATE exam SET grade='A' WHERE avg<=95 AND avg>90;
2 rows updated.

SQL> select * from exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
3333	104	IT	99	82	84	89	100	454	90.8	A
4444	108	IT	92	85	83	91	87	438	87.6	B
5555	106	CSE	82	85	87	91	85	430	86	B
2222	101	IT	98	87	83	99	87	454	90.8	A

Result:

Thus, the various commands in Data Manipulation language was studied and executed successfully.

EX: NO: 2.5 IMPLEMENTATION OF DQL COMMANDS

AIM:

To create a table named "Student" with necessary columns and then perform the DQL Basic commands.

PROCEDURE

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

PROGRAM TO LEARN DQL COMMANDS

CREATING TABLE:

```
SQL> CREATE TABLE student
        (
                studID NUMBER PRIMARY KEY,
                sname VARCHAR2(15) NOT NULL,
                department CHAR(5),
                sem NUMBER,
                dob DATE,
                email_id VARCHAR2(20) UNIQUE,
                college VARCHAR2(10) DEFAULT 'MVIT'
        );
Table created.
```

INSERTING VALUES IN TABLE:
//Inserting values into table student:

```
SQL>INSERT INTO student VALUES(101,'DEEPI','IT',5,'05-JUN-05','deepi@gmail.com','MEC');
1 row created.

SQL> INSERT INTO student VALUES (102,'SHAN','CSE',7,'7-OCT-1995','shan@gmail.com','IIT');
1 row created.

SQL> INSERT INTO student VALUES (104,'REVA','IT',5,'23-JUL-1996','reva@gmail.com','IIT');
1 row created.
```

SQL> INSERT INTO student VALUES (106,'ANANT','CSE',5,'9-JUN-1996', 'anant@gmail.com', 'SMVEC');
1 row created.

SQL> INSERT INTO student VALUES (108,'SASI','IT',5,'21-APR-1996','sasi@gmail.com','MEC');
1 row created.

SELECT ALL COLUMNS:

SQL> select * from student;

STUDID	SNAME	DEPAR	SEM	DOB	EMAIL_ID	COLLEGE
101	DEEPI	IT	5	05-JUN-05	deepi@gmail.com	MEC
102	SHAN	CSE	7	07-OCT-95	shan@gmail.com	IIT
104	REVA	IT	5	23-JUL-96	reva@gmail.com	IIT
106	ANANT	CSE	5	09-JUN-96	anant@gmail.com	SMVEC
108	SASI	IT	5	21-APR-96	sasi@gmail.com	MEC

SELECT MULTICOLUMN:

SQL> SELECT sname, department, sem FROM student;

SNAME	DEPAR	SEM
DEEPI	IT	5
SHAN	CSE	7
REVA	IT	5
ANANT	CSE	5
SASI	IT	5

SELECTION WITH ALIAS COLUMN NAME

SQL> select sname as Sname,department,sem from student;

SNAME	DEPAR	SEM
DEEPI	IT	5
SHAN	CSE	7
REVA	IT	5
ANANT	CSE	5
SASI	IT	5

SELECTION WITH ARITHMETIC OPERATION:

SQL> SELECT sname,department,sem+1 AS "sem" FROM student;

SNAME	DEPAR	sem
DEEPI	IT	6
SHAN	CSE	8
REVA	IT	6
ANANT	CSE	6
SASI	IT	6

DISTINCT RECORD SELECTION:

// To display the college of the student from the table student by avoiding repeated values.

SQL> SELECT DISTINCT college FROM student;

COLLEGE
MEC
IIT
SMVEC

SELECTION WITH CONCATENATION:

SQL> SELECT sname||studid FROM student;

SNAME||STUDENTID
DEEPI101
SHAN102
REVA104
ANANTVI106
SASI108

SELECTION WITH WHERE CLAUSE

// To display the records from the table student who belongs to mec college.

SQL> SELECT * FROM student WHERE college='MEC';

STUDID	SNAME	DEPAR	SEM	DOB	EMAIL_ID	COLLEGE
101	DEEPI	IT	5	05-JUN-05	deepi@gmail.com	MEC
108	SASI	IT	5	21-APR-96	sasi@gmail.com	MEC

// To display the student id and student name from the table student who belongs to iit college

SQL> SELECT studid, sname FROM student WHERE college='IIT';

STUDID	SNAME
102	SHAN
104	REVA

// To display the student name and department from the table student who belongs to 5th sem.

SQL> SELECT sname, department FROM student WHERE sem=5;

SNAME	DEPAR
DEEPI	IT
REVA	IT
ANANT	CSE
SASI	IT

Result:

Thus, the various commands in Data query language was studied and executed successfully.

EX: NO: 2.6 IMPLEMENTATION OF DQL COMMANDS (with constraint)

AIM

To create a table named "Student" with necessary columns and then perform the DQL commands with constraint.

ALGORITHM:

STEP 1: Start.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop.

PROGRAM TO LEARN DQL COMMANDS

CREATING TABLE:

```
SQL> CREATE TABLE student
        (
                studID NUMBER PRIMARY KEY,
                sname VARCHAR2(15) NOT NULL,
                department CHAR(5),
                sem NUMBER,
                dob DATE,
                email_id VARCHAR2(20) UNIQUE,
                college VARCHAR2(10) DEFAULT 'MVIT'
        );
Table created.
```

INSERTING VALUES IN TABLE:
//Inserting values into table student:

```
SQL>INSERT INTO student VALUES(101,'DEEPI','IT',5,'05-JUN-05','deepi@gmail.com','MEC');
1 row created.

SQL> INSERT INTO student VALUES (102,'SHAN','CSE',7,'7-OCT-1995','shan@gmail.com','IIT');
1 row created.


SQL> INSERT INTO student VALUES (104,'REVA','IT',5,'23-JUL-1996','reva@gmail.com','IIT');
1 row created.

SQL> INSERT INTO student VALUES (106,'ANANT','CSE',5,'9-JUN-1996', 'anant@gmail.com',
'SMVEC');
```

1 row created.

SQL> INSERT INTO student VALUES (108,'SASI','IT',5,'21-APR-1996','sasi@gmail.com','MEC');
1 row created.

SELECT ALL COLUMNS:

SQL> select * from student;

STUDID	SNAME	DEPAR	SEM	DOB	EMAIL_ID	COLLEGE
101	DEEPI	IT	5	05-JUN-05	deepi@gmail.com	MEC
102	SHAN	CSE	7	07-OCT-95	shan@gmail.com	IIT
104	REVA	IT	5	23-JUL-96	reva@gmail.com	IIT
106	ANANT	CSE	5	09-JUN-96	anant@gmail.com	SMVEC
108	SASI	IT	5	21-APR-96	sasi@gmail.com	MEC

BETWEEN…AND :

//To display the student id, student name and department of the student whose the semester in between 5 and 6

SQL> SELECT studid,sname,department FROM student WHERE sem BETWEEN 5 AND 6;

STUDID	SNAME	DEPAR
101	DEEPI	IT
104	REVA	IT
106	ANANT	CSE
108	SASI	IT

IN:

// To display the student id, student name and department of the student whose in CSE and IT department

SQL> SELECT studid,sname,department FROM student WHERE department IN ('CSE','IT');

STUDID	SNAME	DEPAR
101	DEEPI	IT
102	SHAN	CSE
104	REVA	IT
106	ANANT	CSE
108	SASI	IT

NOT IN:

// To display the student id, student name and department of the student whose not in CSE department

SQL> SELECT studid,sname,department FROM student WHERE department NOT IN 'CSE';

STUDID	SNAME	DEPAR
101	DEEPI	IT
104	REVA	IT
108	SASI	IT

LIKE:

// To display the student id and student name of the student whose name starts letters 'D' from the table student

SQL> SELECT studid, sname FROM student WHERE sname LIKE 'D%';

STUDID	SNAME
101	DEEPI

// To display the student id and student name of the student whose name end with 'I' from the table student

SQL> SELECT studid, sname FROM student WHERE sname LIKE '%I';

STUDID	SNAME
101	DEEPI
108	SASI

// To display the student id and student name of the student whose name has letters 'E' from the table student

SQL> SELECT studid, sname FROM student WHERE sname LIKE '%E%';

STUDID	SNAME
101	DEEPI
104	REVA

RELATIONAL OPERATOR:

// To display the student id, student name of the student whose the stud id greater than 105

SQL> SELECT studid, sname FROM student WHERE studid > 105;

STUDID	SNAME
106	ANANT
108	SASI

LOGICAL OPERATOR:

// To display the student id, student name of the student whose the student id greater than 105 and cse department

SQL> SELECT studid, sname FROM student WHERE studid > 105 AND department='CSE';

STUDID	SNAME
106	ANANT

Result:

Thus, the various commands in Data query language with constraint was studied and executed successfully.

EX: NO: 2.7 IMPLEMENTATIN OF DCL COMMANDS

AIM

To create a table named "EMP" with necessary columns and then perform the DCL commands.

ALGORITHM

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

The DCL commands are:

- Grant
- Revoke

GRANT:-

Syntax:
Grant <privileges> on <table_name> to user;

Description:

Grant gives specifies SQL statement access or individual data objects to a user or a group of users.

GRANT COMMAND

SQL> grant insert,select,update,delete on emp to system;

Grant succeeded.

REVOKE:-

Syntax:
Revoke <privileges> on <table_name> from user;

Description:

Revoke removes specific SQL statement access previously granted on individual database objects from a user or group of users.

REVOKE COMMAND

```
SQL> revoke select,insert on emp from system;

Revoke succeeded.
```

PROGRAM TO LEARN DCL

CREATE TABLE

```
SQL> create table emp
        (
                eno number primary key,
                ename varchar2(10),
                deptno number,
                deptname varchar2(10)
        );
Table created.
```

//inserting values into table emp

```
SQL> insert into emp values(1, 'suresh', 25, 'IT');
1 row created.
SQL> insert into emp values(2, 'viji', 26, 'CSE');
1 row created.
SQL> insert into emp values(3, 'uthay', 37, 'ECE');
1 row created.
SQL> select * from emp;
```

ENO	ENAME	DEPTNO	DEPTNAME
1	suresh	25	IT
2	viji	26	CSE
3	uthay	37	ECE

```
SQL> GRANT insert, update, select ON emp TO system;
Grant succeeded.
SQL> REVOKE insert,select ON emp FROM system;
Revoke succeeded.
```

Result

Thus, the various commands in Data control language are studied and executed successfully.

EX: NO: 2.8 IMPLEMENTATION OF TCL COMMANDS

AIM

To create a table named "Employee" with necessary columns and then perform the TCL commands.

ALGORITHM

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

The TCL commands are:-

- Commit
- Save point
- Rollback

COMMIT:-

Syntax:

commit;

Description

COMMIT command is used to save the work done.

SAVE POINT:-

Syntax:

Save point pointname;

Description

SAVE POINT command is used to identify a point in a transaction in which it can be restored using Roll back command.

ROLLBACK:-

Syntax:

rollback;

Description

ROLLBACK command is used to restore database to original since last commit.

PROGRAM TO LEARN TCL COMMANDS:
CREATE TABLE EMPLOYEE
//Create table employee

```
SQL> CREATE TABLE employee
        (
                empid NUMBER,
                empname VARCHAR2(15),
                city VARCHAR2(10),
                designation VARCHAR2(15),
                salary NUMBER
        );
```

Table created.

INSERTING VALUES
//Inserting values in table employee

```
SQL> INSERT INTO employee VALUES(100, 'deepi', 'banglore', 'modleader', 20000);
1 row created.
SQL> INSERT INTO employee VALUES(101, 'ganesh', 'chennai', 'engineer', 21000);
1 row created.
SQL> INSERT INTO employee VALUES(102, 'shan', 'calcuta', 'manager', 50000);
1 row created.
SQL> INSERT INTO employee VALUES(103, 'ramki', 'coimbatore', 'proleader', 40000);
1 row created.
SQL> INSERT INTO employee VALUES(104, 'praba', 'bombay', 'ceo', 15000);
1 row created.
SQL> INSERT INTO employee VALUES(105, 'vimal', 'madurai', 'accountant', 15000);
1 row created.
SQL> SELECT * FROM employee;
```

EMPID	EMPNAME	CITY	DESIGNATION	SALARY
100	deepi	banglore	modleader	20000
101	ganesh	chennai	engineer	21000
102	shan	calcuta	manager	50000
103	ramki	Coimbatore	proleader	40000
104	praba	bombay	ceo	15000
105	vimal	madurai	accountant	15000

6 rows selected.

//COMMIT

```
SQL> COMMIT;
Commit complete.
SQL> DELETE FROM employee WHERE empid = 105;
```

1 row deleted.
SQL> SELECT * FROM employee;

EMPID	EMPNAME	CITY	DESIGNATION	SALARY
100	deepi	banglore	modleader	20000
101	ganesh	chennai	engineer	21000
102	shan	calcuta	manager	50000
103	ramki	coimbatore	proleader	40000
104	praba	bombay	ceo	15000

//ROLLBACK

SQL> ROLLBACK;
Rollback complete.

SQL> SELECT * FROM employee;

EMPID	EMPNAME	CITY	DESIGNATION	SALARY
100	deepi	banglore	modleader	20000
101	ganesh	chennai	engineer	21000
102	shan	calcuta	manager	50000
103	ramki	Coimbatore	proleader	40000
104	praba	bombay	ceo	15000
105	vimal	madurai	accountant	15000

6 rows selected.

SQL> DELETE FROM employee WHERE empid = 105;
1 row deleted.

//displaying the values in table employee

SQL> SELECT * FROM employee;

EMPID	EMPNAME	CITY	DESIGNATION	SALARY
100	deepi	banglore	modleader	20000
101	ganesh	chennai	engineer	21000
102	shan	calcuta	manager	50000
103	ramki	coimbatore	proleader	40000
104	praba	bombay	ceo	15000

//COMMIT
SQL> COMMIT;

Commit complete.

//ROLLBACK
SQL> ROLLBACK;
Rollback complete.

//displaying values in the table employee

SQL> SELECT * FROM employee;

EMPID	EMPNAME	CITY	DESIGNATION	SALARY
100	deepi	banglore	modleader	20000
101	ganesh	chennai	engineer	21000
102	shan	calcuta	manager	50000
103	ramki	coimbatore	proleader	40000
104	praba	bombay	ceo	15000

//Creating Savepoint

SQL> SAVEPOINT s1;
Savepoint created.

SQL> UPDATE employee SET salary = 50000 WHERE empid = 104 ;
1 row updated.

SQL> SAVEPOINT s2;
Savepoint created.

SQL> UPDATE employee SET salary = 45000 WHERE empid = 103;
1 row updated.

SQL> SELECT * FROM employee;

EMPID	EMPNAME	CITY	DESIGNATION	SALARY
100	deepi	banglore	modleader	20000
101	ganesh	chennai	engineer	21000
102	shan	calcuta	manager	50000
103	ramki	coimbatore	proleader	45000
104	praba	bombay	ceo	50000

// rollback savepoint s2

SQL> ROLLBACK to s2;

Rollback complete.

SQL> SELECT * FROM employee;

EMPID	EMPNAME	CITY	DESIGNATION	SALARY
100	deepi	banglore	modleader	20000
101	ganesh	chennai	engineer	21000
102	shan	calcuta	manager	50000
103	ramki	coimbatore	proleader	40000
104	praba	bombay	ceo	50000

Result

Thus, the various commands in Transaction control language are studied and executed successfully.

EX: NO: 2.9 BUILT-IN-FUNCTIONS

AIM

To execute and verify the various BUILT IN FUNCTIONS.

PROCEDURE

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

NUMERIC FUNCTIONS

- ABS
- POWER
- ROUND
- TRUNC
- SQRT
- FLOOR
- CEIL

ABS:-

Syntax:

Select abs(field1) from <tablename>;

Description

Returns the absolute value of 'n'.

POWER:-

Syntax:

Select power(field1,field2) from <tablename>;

Description

Returns m raised to the n^{th} power ,n must be an integer else an error is returned.

ROUND:-

Syntax:

Select round(field1,[field2]) from <tablename>;

Description

Returns n,rounded to the places to the right of the decimal point.

SQRT:-

Syntax:
Select sqrt(field1) from <tablename>;

Description

Returns the square root of n.

TRUNC:-

Syntax:
Select trunc(field1,[field2]) from <tablename>;

Description

Returns a number truncated to a certain number of decimal places. If this parameter is omitted, the TRUNC function will truncate the number to 0 decimal places.

FLOOR:-

Syntax:
Select floor(field1) from <tablename>;

Description

Returns the largest integer value that is equal to or less then a number.

CEIL:-

Syntax:
Select ceil(field1) from <tablename>;

Description

Returns the smallest integer value that is greater than or equal to a number.

STRING FUNCTIONS

- LOWER
- UPPER
- ASCII

LOWER:-

Syntax:
Select Lower(field1) from <tablename>;

Description

Returns char with all letters in lowercase.

UPPER:-

Syntax:
Select Upper(field1) from <tablename>;

Description

Returns char with all letters forced to uppercase.

ASCII:-

Syntax:
Select ASCII(char) from <tablename>;

Description

Returns the number code that represents the specified character.

COMMAND EXECUTION

NUMERIC FUNCTIONS:

```
SQL> select abs(-99) from dual;

  ABS(-99)
----------
        99

SQL> select abs(45.95) from dual;

  ABS(45.95)
-------------
       45.95
SQL> select power(3,2) from dual;

  POWER(3,2)
------------
           9
SQL> select SQRT(625) from dual;

  SQRT(625)
-----------
          25
SQL> select SQRT(144) from dual;

  SQRT(144)
-----------
          12
SQL> select CEIL(22.22) from dual;

  CEIL(22.22)
--------------
```

```
        23
SQL> select FLOOR(45.49) from dual;

FLOOR(45.49)
------------
        45
SQL> select ROUND(66.66) from dual;

ROUND(66.66)
------------
        67
SQL> select TRUNC(55.99) from dual;

TRUNC(55.99)
------------
        55
SQL> select LN(2) from dual;

     LN(2)
----------
.693147181
SQL> select LN(55) from dual;

    LN(55)
----------
4.00733319
SQL> select SIN(60) from dual;

   SIN(60)
----------
-.30481062
SQL> select SIN(0) from dual;

    SIN(0)
----------
         0
SQL> select cos(0) from dual;

    COS(0)
----------
         1
SQL> select TAN(30) from dual;

   TAN(30)
----------
-6.4053312
```

STRING FUNCTIONS

```
SQL> select * from dual;

D
-
X
SQL> select ascii('A') from dual;

ASCII('A')
----------
        65
SQL> select ascii('ab') from dual;

ASCII('AB')
-----------
         97
SQL> select ascii('R') from dual;

ASCII('R')
----------
        82
SQL> select lower('SAGAR') from dual;

LOWER
-----
sagar
SQL> select upper('sagar') from dual;

UPPER
-----
SAGAR
SQL> spool off
not spooling currently
```

Result

Thus the Built-in functions has been verified and executed successfully.

EX: NO: 2.10 AGGREGATE FUNCTIONS

AIM

To create a table named "Student and Exam" with necessary columns and then perform the Aggregate functions.

ALGORITHM:

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

Built-In Functions are (Aggregate):

- MAX
- MIN
- COUNT
- SUM
- AVG

1. MAX:-

Syntax:
Select max(field1) from <table_name>;

Description

MAX command is used to find the maximum among the entities in a particular attribute.

2. MIN:-

Syntax:
Select min(field1) from <table_name>;

Description

MIN command is used to find the minimum among the entities in a particular attribute.

3. COUNT:-

Syntax:
Select count(field1) from <table_name>;

Description
COUNT command is used to count the entire entities in a particular attribute.

4. SUM:-

Syntax:
Select sum(field1) from <table_name>;

Description

SUM command is used to add all the entities with in the attribute.

5. AVG:-

Syntax:
Select avg(field1) from <table_name>;

Description

AVG command is used to find average of entity in particular attribute.

PROGRAM TO LEARN AGGREGATE FUNCTION:

CREATING TABLE:

// To create the Table student:

```
SQL> CREATE TABLE student
        (
                sID NUMBER PRIMARY KEY,
                sname VARCHAR2(15) NOT NULL,
                department CHAR(5),
                sem NUMBER,
                dob DATE,
                email_id VARCHAR2(20) UNIQUE,
                college VARCHAR2(10) DEFAULT 'MVIT'
        );
```

Table created.

INSERTING VALUES IN TABLE:

//insert values into student table:

```
SQL>INSERT INTO student VALUES(101,'DEEPI','IT',5,'05-JUN-05','deepi@gmail.com','MEC');
1 row created.
SQL> INSERT INTO student VALUES (102,'SHAN','CSE',7,'7-OCT-1995','shan@gmail.com','IIT');
1 row created.

SQL> INSERT INTO student VALUES (104,'REVA','IT',5,'23-JUL-1996','reva@gmail.com','IIT');
1 row created.

SQL> INSERT INTO student VALUES (106,'ANANT','CSE',5,'9-JUN-1996', 'anant@gmail.com',
'SMVEC');
1 row created.

SQL> INSERT INTO student VALUES (108,'SASI','IT',5,'21-APR-1996','sasi@gmail.com','MEC');
1 row created.

SQL>   select * from student;
```

SID	SNAME	DEPAR	SEM	DOB	EMAIL_ID	COLLEGE
101	DEEPI	IT	5	05-JUN-05	deepi@gmail.com	MEC
102	SHAN	CSE	7	07-OCT-95	shan@gmail.com	IIT
104	REVA	IT	5	23-JUL-96	reva@gmail.com	IIT
106	ANANT	CSE	5	09-JUN-96	anant@gmail.com	SMVEC
108	SASI	IT	5	21-APR-96	sasi@gmail.com	MEC

// To create a table exam

```
SQL> CREATE TABLE exam
        (
                EID NUMBER PRIMARY KEY,
                SID NUMBER REFERENCES student(SID),
                dept CHAR(5) NOT NULL,
                m1 NUMBER CHECK (m1<=100 and m1>=0),
                m2 NUMBER CHECK (m2<=100 and m2>=0),
                m3 NUMBER CHECK (m3<=100 and m3>=0),
                m4 NUMBER CHECK (m4<=100 and m4>=0),
                m5 NUMBER CHECK (m5<=100 and m5>=0),
                tot NUMBER,
                avg NUMBER,
                grade CHAR(1)
        );
 Table created.
```

//insert value into exam table:

SQL> INSERT INTO exam(eid, sid, dept, m1, m2, m3, m4, m5)VALUES (2222,101,'IT',98,87,83,99,87);
1 row created.
SQL> INSERT INTO exam(eid, sid, dept, m1, m2, m3, m4,m5)VALUES(3333,104,'IT',99,82,84,89,100);
1 row created.
SQL> INSERT INTO exam(eid, sid, dept, m1, m2, m3, m4,m5)VALUES(4444,108,'IT',92,85,83,91,87);
1 row created.
SQL> INSERT INTO exam(eid, sid,dept, m1, m2, m3, m4,m5)VALUES(5555,106,'CSE',82,85,87,91,85);
1 row created.

SQL> select * from exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
2222	101	IT	98	87	83	99	87			
3333	104	IT	99	82	84	89	100			
4444	108	IT	92	85	83	91	87			
5555	106	CSE	82	85	87	91	85			

// To set the total in the table exam

SQL> UPDATE exam SET tot=(m1+m2+m3+m4+m5);
4 rows updated.
SQL> select * from exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
2222	101	IT	98	87	83	99	87	454		
3333	104	IT	99	82	84	89	100	454		
4444	108	IT	92	85	83	91	87	438		
5555	106	CSE	82	85	87	91	85	430		

// To set the average in the table exam

SQL> UPDATE exam SET avg=tot/5;
4 rows updated.
SQL> select * from exam;

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
2222	101	IT	98	87	83	99	87	454	90.8	
3333	104	IT	99	82	84	89	100	454	90.8	
4444	108	IT	92	85	83	91	87	438	87.6	
5555	106	CSE	82	85	87	91	85	430	86	

// To set the grade in the table exam

SQL> UPDATE exam SET grade='S' WHERE avg>95;
0 rows updated.
SQL> UPDATE exam SET grade='A' WHERE avg<=95 AND avg>90;
2 rows updated.

```
SQL> UPDATE exam SET grade='B' WHERE avg<=90 AND avg>85;
2 rows updated.

SQL> UPDATE exam SET grade='C' WHERE avg<=85 AND avg>80;
0 rows updated.
SQL> UPDATE exam SET grade='D' WHERE avg<=80 AND avg>75;
0 rows updated.
SQL> UPDATE exam SET grade='F' WHERE avg<75;
0 rows updated.
  SQL> select * from exam;
```

EID	SID	DEPT	M1	M2	M3	M4	M5	TOT	AVG	GRADE
2222	101	IT	98	87	83	99	87	454	90.8	A
3333	104	IT	99	82	84	89	100	454	90.8	A
4444	108	IT	92	85	83	91	87	438	87.6	B
5555	106	CSE	82	85	87	91	85	430	86	B

ORDER BY:

//To display the department, sem and student name from the table student based on department in ascending order.

```
SQL> SELECT department, sem, sname FROM student ORDER BY department;
```

DEPAR	SEM	SNAME
CSE	5	ANANT
CSE	7	SHAN
IT	5	SASI
IT	5	DEEPI
IT	5	REVA

//To display the department, sem and student name from the table student based on department in descending order.

```
SQL> SELECT department, sem, sname FROM student ORDER BY department DESC, sem DESC, sname
DESC;
```

DEPAR	SEM	SNAME
IT	5	DEEPI
IT	5	SASI
IT	5	REVA
CSE	7	SHAN
CSE	5	ANANT

GROUP BY:

// To displays the total value group by department

```
SQL> SELECT department, SUM(tot) AS SUM_DEPARTMENT FROM exam GROUP BY
department;
```

DEPAR SUM_DEPARTMENT

IT 1346
CSE 430

AGGREGATE FUNCTIONS:

// 1. COUNT - displays total number of rows:

SQL> SELECT COUNT (eid) AS STUDENTS_REGISTERED FROM exam;

STUDENTS_REGISTERED

4

// 2. MAX - displays the maximum value:

SQL> SELECT MAX(avg) AS RANK_1 FROM exam;

RANK_1

90.8

// 3. MIN - displays the minimum value

SQL> SELECT MIN(avg) AS LAST_RANK FROM exam;

LAST_RANK

86

// 4. SUM - displays the total value:

SQL> SELECT department, SUM(tot) AS SUM_DEPARTMENT FROM exam5 GROUP BY department;

DEPAR SUM_DEPARTMENT

IT 1346
CSE 430

// 5. AVG-display average of total

SQL> select avg(tot) from exam;

AVG(TOT)

444

Result:

Thus, the various commands in aggregate functions was studied and executed successfully.

EX: NO: 3.1 SET OPERATIONS

AIM

To create a table named "Employee" with necessary columns and then perform the Set Operations.

ALGORITHM:

STEP 1: Start.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop.

Set Operations Are:

- UNION
- UNIONALL
- INTERSECT
- MINUS

UNION:-

Syntax:
Select <fieldname1> from <table_name1> union Select <fieldname2> from <table_name2>;

Description
UNION command is used to compile all distinct rows and display all entities in both rows.

UNIONALL:-

Syntax:
Select <fieldname1> from <table_name1> unionall Select <fieldname2> from <table_name2>;

Description
UNIONALL command is used to return all entities in both rows.

INTERSECT:-

Syntax:
Select <fieldname1> from <table_name1> intersect Select <fieldname2> from <table_name2>;

Description

INTERSECT command is used to display only similar entities in both rows.

MINUS:-

Syntax:
Select <fieldname1> from <table_name1> minus Select <fieldname2> from <table_name2>;

Description

MINUS command is used to display only the rows that don't match in both queries.

PROGRAM TO LEARN SET OPERATION:

CREATING TABLE

//Create Table employee

```
SQL> CREATE TABLE employee
        (
                Employee_Name VARCHAR2(10),
                Employee_no NUMBER PRIMARY KEY,
                Dept_no NUMBER,
                Dept_name VARCHAR2(10)
                );
Table created.
```

//Create Table Employee1

```
SQL> CREATE TABLE employee1
        (
                Employee_Name VARCHAR2(10),
                Employee_no NUMBER PRIMARY KEY,
                Dept_no NUMBER,
                 dept_name VARCHAR2(10)
        );
Table created.
```

//Inserting Values into table employee

```
SQL> INSERT INTO employee VALUES('Ganesh', 234, 45, 'CSE');
1 row created.

SQL> INSERT INTO employee VALUES('Vijay',  877, 85, 'EEE');
1 row created.
```

SQL> INSERT INTO employee VALUES('Vignesh', 990, 95, 'BME');
1 row created.

//Inserting Values into table employee1

SQL> INSERT INTO employee1 VALUES('Ganesh', 234, 45, 'CSE');
1 row created.

SQL> INSERT INTO employee1 VALUES('Vishnu', 476, 55, 'IT');
1 row created.

SQL> INSERT INTO employee1 VALUES('Vikram', 985, 75, 'ECE');
1 row created.

//Displaying the values in the table employee

SQL> SELECT * FROM employee;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NO	DEPT_NAME
Ganesh	234	45	CSE
Vijay	877	85	EEE
Vignesh	990	95	BME

//Displaying the values in the table employee1

SQL> SELECT * FROM employee1;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NO	DEPT_NAME
Ganesh	234	45	CSE
Vishnu	476	55	IT
Vikram	985	75	ECE

UNION:
//To display employee_no using union - displays all values without duplicates from employee and employee1

SQL> SELECT employee_no FROM employee UNION SELECT employee_no FROM employee1;

EMPLOYEE_NO
234
476

877
985
990

UNION ALL:
//To display employee_no using union all - displays all values with duplicates from employee and employee1

SQL> SELECT employee_no FROM employee UNION ALL SELECT employee_no FROM employee1;

EMPLOYEE_NO

234
877
990
234
476
985

6 rows selected.

INTERSECT:
//To display employee_no using intersect - displays common values in both the tables without duplicates from employee and employee1

SQL> SELECT employee_no FROM employee INTERSECT SELECT employee_no FROM employee1;

EMPLOYEE_NO

234

MINUS:
//To display employee_no using minus - displays the values that minus the employee_no of employee from employee1

SQL> SELECT employee_no FROM employee MINUS SELECT employee_no FROM employee1;

EMPLOYEE_NO

877
990

Result:

Thus, the various commands in Set operations was studied and executed successfully.

EX: NO: 3.2 JOINS

AIM

To create a table named "Cseitstudent and Placement" with necessary columns and then perform the Joins.

ALGORITHM

STEP 1: Start

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop

DEFINITION AND SYNTAX:

JOINS

The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

```
SELECT *
FROM table_name1
[JOIN KEYWORD] table_name2
WHERE table_name1.column = table_name2.column
```

1. INNER JOIN

Returns all the attributes from both tables including repeated attributes based on condition

2. OUTER JOIN

A. Left Outer Join

Return all rows FROM the left table, even if there are no match in the right table

B. Right Outer Join

Return all rows FROM the right table, even if there are no match in the left table

C. Full Outer Join

Returns all the rows FROM both the tables.

3. SELF JOIN

Returns rows by comparing the values of the same table.

4. EQUI JOIN

Returns the rows FROM two tables that satisfies the equal to condition

5. NON EQUI JOIN

Returns the rows FROM two tables that satisfies the non equal to condition

PROGRAM TO LEARN JOINS
CREATE A TABLE:
//To create a table cseitstudent

```
SQL> CREATE TABLE cseitstudent
        (
                studID NUMBER PRIMARY KEY,
                sname VARCHAR(15),
                dept CHAR(5),
                sem NUMBER
        );
Table created.
```

//To create a table placement

```
SQL> CREATE TABLE placement
        (
                PID NUMBER PRIMARY KEY,
                StudID NUMBER,
                dept CHAR(5),
                Company VARCHAR2(30),
                salary NUMBER
        );
Table created.
```

INSERTING VALUES IN cseitstudent TABLE:

```
SQL> INSERT INTO cseitstudent VALUES(101,'deepi', 'IT',5);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(102,'reva', 'IT',3);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(103,'navya', 'CSE',3);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(104,anant', 'IT',3);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(105,'arnav', 'CSE',5);
1 row created.
```

INSERTING VALUES IN placement TABLE:

```
SQL> INSERT INTO placement VALUES(1, 104, 'IT', 'infosys', 25000);
1 row created.
```

```
SQL> INSERT INTO placement VALUES(2, 105, 'CSE', 'Wipro', 22000);
1 row created.
SQL> INSERT INTO placement VALUES(3, 204, 'MECH', 'HYUNDAI', 30000);
1 row created.
SQL> INSERT INTO placement VALUES(4, 102, 'IT', 'infosys', 25000);
1 row created.
SQL> INSERT INTO placement VALUES(5, 103, 'CSE', 'infosys', 25000);
1 row created.
```

DESCRIBING THE TABLE

// Display the values in the table as rows

```
SQL> SELECT * FROM cseitstudent;
```

STUDID	SNAME	DEPT	SEM
101	deepi	IT	5
102	reva	IT	3
103	navya	CSE	3
104	anant	IT	3
105	arnav	CSE	5

```
SQL> SELECT * FROM placement;
```

PID	STUDID	DEPT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	HYUNDAI	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000

INNER JOIN

```
SQL> SELECT * FROM cseitstudent INNER JOIN Placement ON
cseitstudent.studID=placement.StudID;
```

STUDID	SNAME	DEPT	SEM	PID	STUDID	DEPT	COMPANY
104	anant	IT	3	1	104	IT	infosys
105	arnav	CSE	5	2	105	CSE	Wipro
102	reva	IT	3	4	102	IT	infosys
103	navya	CSE	3	5	103	CSE	infosys

```
SQL> SELECT cseitstudent.studID, cseitstudent.sname, placement.company,   placement.salary
        FROM cseitstudent   INNER JOIN placement ON cseitstudent.studID=placement.studID;
```

STUDID	SNAME	COMPANY	SALARY
104	anant	infosys	25000
105	arnav	Wipro	22000

102	reva	infosys	25000
103	navya	infosys	25000

LEFT OUTER JOIN:

SQL> SELECT * FROM cseitstudent LEFT OUTER JOIN placement ON cseitstudent.studID=placement.studID;

STUDID	SNAME	DEPT	SEM	PID	STUDID	DEPT	COMPANY
101	deepi	IT	5				
102	reva	IT	3	4	102	IT	infosys
103	navya	CSE	3	5	103	CSE	infosys
104	anant	IT	3	1	104	IT	infosys
105	arnav	CSE	5	2	105	CSE	Wipro

SQL> SELECT cseitstudent.sname,placement.pID, placement.company FROM cseitstudent LEFT OUTER JOIN placement ON cseitstudent.studID=placement.studID;

SNAME	PID	COMPANY
deepi		
reva	4	infosys
navya	5	infosys
anant	1	infosys
arnav	2	Wipro

RIGHT OUTER JOIN:

SQL> SELECT * FROM cseitstudent RIGHT OUTER JOIN placement ON cseitstudent.studID=placement.studID;

STUDID	SNAME	DEPT	SEM	PID	STUDID	DEPT	COMPANY
104	anant	IT	3	1	104	IT	infosys
105	arnav	CSE	5	2	105	CSE	wipro
				3	204	MECH	hyundai
102	reva	IT	3	4	102	IT	infosys
103	navya	CSE	3	5	103	CSE	infosys

SQL> SELECT cseitstudent.sname,placement.pID, placement.company FROM cseitstudent RIGHT OUTER JOIN placement ON cseitstudent.studID = placement.studID;

SNAME	PID	COMPANY
Anant	1	infosys
Arnav	2	wipro
	3	hyundai
Reva	4	infosys

Navya 5 infosys

FULL OUTER JOIN:

```
SQL> SELECT * FROM cseitstudent  FULL OUTER JOIN placement
        ON cseitstudent.studID=placement.studID;
```

STUDID	SNAME	DEPT	SEM	PID	STUDID	DEPT	COMPANY
101	deepi	IT	5				
102	reva	IT	3	4	102	IT	infosys
103	navya	CSE	3	5	103	CSE	infosys
104	anant	IT	3	1	104	IT	infosys
105	arnav	CSE	5	2	105	CSE	wipro
				3	204	MECH	hyundai

6 rows selected.

```
SQL> SELECT cseitstudent.sname,placement.pID, placement.company FROM cseitstudent
        FULL OUTER JOIN placement  ON cseitstudent.studID=placement.studID;
```

SNAME	PID	COMPANY
deepi		
reva	4	infosys
sheela	5	infosys
anant	1	infosys
arnav	2	wipro
	3	hyundai

6 rows selected.

TABLE CREATION:

```
// to create the table employee
SQL> CREATE TABLE employee
        (
                empid NUMBER,
                empname VARCHAR2(25),
                reportingid NUMBER
        );
```

Table created.

INSERTING VALUES IN TABLE:

```
SQL>INSERT INTO employee VALUES(1, 'Principal', null);
1 row created.
SQL>INSERT INTO employee VALUES(2, 'HOD', 1);
1 row created.
SQL>INSERT INTO employee VALUES(3, 'PO', 1);
1 row created.
SQL>INSERT INTO employee VALUES(4, 'Staff', 2);
1 row created.
SQL>INSERT INTO employee VALUES(5, 'Non Teaching Staff', 2);
1 row created.
```

//describing the table

SQL> select * from employee;

EMPID	EMPNAME	REPORTINGID
1	Principal	
2	HOD	1
3	PO	1
4	Staff	2
5	Non Teaching Staff	2

SELF JOIN:

SQL> SELECT e1.empid, e1.empname, e2.empname AS HeadName FROM employee e1, employee e2 WHERE e1.reportingid=e2.empid;

EMPID	EMPNAME	HEADNAME
3	PO	Principal
2	HOD	Principal
5	Non Teaching Staff	HOD
4	Staff	HOD

EQUI JOIN:

SQL> SELECT * FROM cseitstudent, placement WHERE cseitstudent.studID=placement.studID;

STUDID	SNAME	DEPT	SEM	PID	STUDID	DEPT	COMPANY
104	anant	IT	3	1	104	IT	infosys
105	arnav	CSE	5	2	105	CSE	wipro
102	reva	IT	3	4	102	IT	infosys
103	navya	CSE	3	5	103	CSE	infosys

NON EQUI JOIN:

SQL> SELECT cseit.studID, cseit.sname FROM cseitstudent cseit, placement placed
WHERE cseit.studID>placed.studID;

STUDID	SNAME
105	arnav
103	navya
104	anant
105	arnav
104	anant
105	arnav

6 rows selected.

RESULT:

Thus the Various commands in joins are studied and executed Successfully.

EX: NO: 3.3 NESTED SUB QUREIES

AIM

To create a table named "Employee1, Cseitstudent and Placement" with necessary columns and then perform the Nested Sub Queries.

ALGORITHM:

STEP 1: Start.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop.

PROGRAM TO LEARN NESTED SUB QUERIES

CREATING TABLE:

```
SQL> CREATE TABLE employee1
        (
                empno NUMBER,
                empname VARCHAR2(15),
                salary NUMBER,
                designation VARCHAR2(10)
        );
Table created.
```

INSERTING VALUES INTO TABLE EMPLOYEE1:

```
SQL> INSERT INTO employee1  VALUES(100,'DEEPI',20000,'Clerk');
1 row created.
SQL> INSERT INTO employee1 VALUES(101,'SHAN',10000,'Typist');
1 row created.
SQL> INSERT INTO employee1 VALUES(102,'SHOLK',15000,'Cashier');
1 row created.
```

DISPLAYING THE VALUES IN TABLE EMPLOYEE1:

```
SQL> select * from employee1;
```

EMPNO	EMPNAME	SALARY	DESIGNATIO
100	DEEPI	20000	Clerk
101	SHAN	10000	Typist
102	SHOLK	15000	Cashier

EXAMPLE QUERIES:

SQL> SELECT * FROM employee1 WHERE designation = (SELECT designation FROM employee1 WHERE empname='RUPESH');

EMPNO	EMPNAME	SALARY	DESIGNATIO
100	DEEPI	20000	Clerk

SQL> SELECT * FROM employee1 WHERE salary < (SELECT salary FROM employee1 WHERE empname='RUPESH');

EMPNO	EMPNAME	SALARY	DESIGNATIO
101	SHAN	10000	Typist
102	SHOLK	15000	Cashier

CREATE TABLE:

//To create a table cseitstudent

```
SQL> CREATE TABLE cseitstudent
        (
                studentID NUMBER PRIMARY KEY,
                sname VARCHAR(15),
                department CHAR(5),
                sem NUMBER
        );
Table created.
```

//To create a table placement

```
SQL> CREATE TABLE placement
        (
                PlacementID NUMBER PRIMARY KEY,
                StudentID NUMBER,
                department CHAR(5),
                Company VARCHAR2(30),
                salary NUMBER
        );
Table created.
```

INSERTING VALUES IN TABLE:

// Inserting values into cseitstudent table

```
SQL> INSERT INTO cseitstudent VALUES(101,'ranvir', 'IT',5);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(102,'rajesh', 'IT',3);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(103,'sasi', 'CSE',3);
1 row created.
```

SQL> INSERT INTO cseitstudent VALUES(104,'sidhu', 'IT',3);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(105,'astha', 'CSE',5);
1 row created.
// Inserting values into placement table

SQL> INSERT INTO placement VALUES(1, 104, 'IT', 'infosys', 25000);
1 row created.
SQL> INSERT INTO placement VALUES(2, 105, 'CSE', 'Wipro', 22000);
1 row created.
SQL> INSERT INTO placement VALUES(3, 204, 'MECH', 'HYUNDAI', 30000);
1 row created.
SQL> INSERT INTO placement VALUES(4, 102, 'IT', 'infosys', 25000);
1 row created.
SQL> INSERT INTO placement VALUES(5, 103, 'CSE', 'infosys', 25000);
1 row created.
DESCRIBING THE TABLE
// Display the values in the table as rows
SQL> SELECT * FROM cseitstudent;

STUDENTID	SNAME	DEPAR	SEM
101	ranvir	IT	5
102	rajesh	IT	3
103	sasi	CSE	3
104	sidhu	IT	3
105	astha	CSE	5

SQL> SELECT * FROM placement;

PLACEMENTID	STUDENTID	DEPAR	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	HYUNDAI	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000

EXAMPLE QUERIES:

SQL> SELECT studentID, sname FROM cseitstudent WHERE studentID IN
(SELECT studentID FROM placement);

STUDENTID	SNAME
102	rajesh
103	sasi
104	sidhu
105	astha

SQL> SELECT studentID, sname FROM cseitstudent WHERE studentID NOT IN

(SELECT studentID FROM placement);

STUDENTID	SNAME
101	ranvir

SQL> SELECT sname, department FROM cseitstudent WHERE studentID IN
(SELECT studentID FROM placement WHERE company='infosys');

SNAME	DEPAR
rajesh	IT
sasi	CSE
sidhu	IT

SQL> SELECT sname, department FROM cseitstudent WHERE studentID NOT IN
(SELECT studentID FROM placement WHERE company='infosys');

SNAME	DEPAR
ranvir	IT
astha	CSE

SQL> SELECT studentID, company, salary FROM placement WHERE
SALARY < SOME (23000, 28000);

STUDENTID	COMPANY	SALARY
104	infosys	25000
105	Wipro	22000
102	infosys	25000
103	infosys	25000

SQL> SELECT studentID, company, salary FROM placement WHERE Salary > SOME
(SELECT salary FROM placement WHERE company='infosys');

STUDENTID	COMPANY	SALARY
204	HYUNDAI	30000

SQL> SELECT studentID, company, salary FROM placement WHERE salary < ALL(23000,28000);

STUDENTID	COMPANY	SALARY
105	Wipro	22000

SQL> SELECT studentID, company, salary FROM placement WHERE salary > ALL
(SELECT salary FROM placement WHERE company='infosys');

STUDENTID	COMPANY	SALARY
204	HYUNDAI	30000

EXISTS:

SQL> SELECT * FROM employee1 WHERE EXISTS (SELECT * FROM employee1 WHERE salary>5000);

EMPNO	EMPNAME	SALARY	DESIGNATION
100	DEEPI	20000	Clerk
101	SHAN	10000	Typist
102	SHOLK	15000	Cashier

SQL> SELECT * FROM placement WHERE EXISTS (SELECT * FROM placement WHERE salary>20000);

PLACEMENTID	STUDENTID	DEPAR	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	HYUNDAI	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000

NOT EXISTS

SQL> SELECT * FROM employee1 WHERE NOT EXISTS (SELECT * FROM employee1 WHERE salary>5000);
no rows selected

RESULT:

Thus, the various NESTED SUB QUEIRIES commands was studied and executed successfully.

EX: NO: 3.4 VIEWS

AIM

To create a table named " Cseitstudent and Placement" with necessary columns and then perform and update View in RDBMS.

ALGORITHM:

STEP 1: Start.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Execute different Commands and extract information from the table.

STEP 5: Stop.

DEFINITION AND SYNTAX:

VIEW

A view is a virtual table based on the result-set of an SQL statement.A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

Creating a View
Syntax:
CREATE VIEW view_name AS SELECT column_name(s) FROM table_name
WHERE condition

Updating a View
Syntax:
CREATE OR REPLACE VIEW view_name AS SELECT column_name(s) FROM table_name
WHERE condition

Dropping a View
Syntax:
DROP VIEW view_name

PROGRAM TO LEARN VIEWS
CREATE A TABLE:
//To create a table cseitstudent

```
SQL> CREATE TABLE cseitstudent
        (
                studentID NUMBER PRIMARY KEY,
                sname VARCHAR(15),
                department CHAR(5),
                sem NUMBER
        );
Table created.
```

//To create a table placement

```
SQL> CREATE TABLE placement
        (
                PlacementID NUMBER PRIMARY KEY,
                StudentID NUMBER,
                departmentt CHAR(5),
                Company VARCHAR2(30),
                salary NUMBER
        );
Table created.
```

INSERTING VALUES IN TABLE:

// Inserting values into cseitstudent table

```
SQL> INSERT INTO cseitstudent VALUES(101,'reema', 'IT',5);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(102,'reenu', 'IT',3);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(103,'sheela', 'CSE',3);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(104,nirmal', 'IT',3);
1 row created.
SQL> INSERT INTO cseitstudent VALUES(105,'eshwar', 'CSE',5);
1 row created.
```

// Inserting values into placement table

```
SQL> INSERT INTO placement VALUES(1, 104, 'IT', 'infosys', 25000);
1 row created.
SQL> INSERT INTO placement VALUES(2, 105, 'CSE', 'Wipro', 22000);
1 row created.
SQL> INSERT INTO placement VALUES(3, 204, 'MECH', 'HYUNDAI', 30000);
1 row created.
SQL> INSERT INTO placement VALUES(4, 102, 'IT', 'infosys', 25000);
1 row created.
SQL> INSERT INTO placement VALUES(5, 103, 'CSE', 'infosys', 25000);
1 row created.
```

//displaying the values

```
SQL> select * from cseitstudent;
```

STUDENTID	SNAME	DEPARTMENT	SEM
101	reema	IT	5
102	reenu	IT	3
103	sheela	CSE	3
104	nirmal	IT	3
105	eshwar	CSE	5

```
SQL> select * from placement;
```

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	Hyundai	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000

//Consider the placement table
SELECT * FROM placement;

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	Hyundai	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000

// Simple Example for View
// View creation
CREATE VIEW studview AS(SELECT studentid,department,company FROM placement WHERE salary >25000);

//To displat the records in studview
SELECT * FROM studview;

STUDENTID	DEPAR	COMPANY
204	MECH	HYUNDAI

// **To create a view by using inner join of cseitstudent and placement**
CREATE VIEW studetails AS(SELECT
cseitstudent.studentID,cseitstudent.sname,cseitstudent.department,
cseitstudent.sem,placement.company,placement.salary FROM cseitstudent
INNER JOIN placement ON cseitstudent.studentID=placement.studentID);

// To display the records in studetails view

SELECT * FROM studetails;

STUDENTID	SNAME	DEPARTMENT	SEM	COMPANY	SALARY
104	nirmal	IT	3	infosys	25000
105	eshwar	CSE	5	Wipro	22000
102	reenu	IT	3	infosys	25000
103	sheela	CSE	3	infosys	25000

// To insert a new record in base table of cseitstudent table

```
INSERT INTO cseitstudent(studentID,sname,department,sem) VALUES(107,'kannan','CSE',5);
```

// To insert a new record in base table of placement table

```
INSERT INTO placement(placementID,studentID,department,company,salary)
VALUES(6,107,'CSE','infosys',25000);
```

// To display the records in cseitstudent table (After inserted a new record)

```
SELECT * FROM cseitstudent;
```

STUDENTID	SNAME	DEPARTMENT	SEM
101	reema	IT	5
102	reenu	IT	3
103	sheela	CSE	3
104	nirmal	IT	3
105	eshwar	CSE	5
106	kannan	CSE	5

//To display the records in placement table (After inserted a new record)

```
SELECT * FROM placement;
```

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	Hyundai	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000
6	106	CSE	infosys	25000

//To display the records in placement table (After inserted a new record in base tables)

```
SELECT * FROM studetails;
```

STUDENTID	SNAME	DEPARTMENT	SEM	COMPANY	SALARY
104	nirmal	IT	3	infosys	25000
105	eshwar	CSE	5	Wipro	22000
102	reenu	IT	3	infosys	25000
103	sheela	CSE	3	infosys	25000
106	kannan	CSE	5	infosys	25000

// To delete a particular record in studetails View

```
DELETE FROM studetails WHERE studentid=106;
```

// To display the records in studetails view (After deleted a new record in view)

```
SELECT * FROM studetails;
```

STUDENTID	SNAME	DEPARTMENT	SEM	COMPANY	SALARY
104	nirmal	IT	3	infosys	25000

105	eshwar	CSE	5	Wipro	22000
102	reenu	IT	3	infosys	25000
103	sheela	CSE	3	infosys	25000

// To display the records in placement table (After deleted a new record in view)

SELECT * FROM placement;

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	Hyundai	30000
4	102	IT	infosys	25000
5	103	CSE	infosys	25000

//To update a particular record in studetails view

UPDATE studetails SET salary=27500 WHERE studentid=102;

// To display the records in studetails view (After updated a new record in view)

SELECT * FROM studetails;

STUDENTID	SNAME	DEPARTMENT	SEM	COMPANY	SALARY
104	nirmal	IT	3	infosys	25000
105	eshwar	CSE	5	Wipro	22000
102	reenu	IT	3	infosys	27500
103	sheela	CSE	3	infosys	25000

// To display the records in placement table (After deleted a new record in view)

SELECT * FROM placement;

PLACEMENTID	STUDENTID	DEPARTMENT	COMPANY	SALARY
1	104	IT	infosys	25000
2	105	CSE	Wipro	22000
3	204	MECH	Hyundai	30000
4	102	IT	infosys	27500
5	103	CSE	infosys	25000

SQL> DROP VIEW studetails;
View dropped.
SQL> SELECT * FROM studetails;
ERROR at line 1:
ORA-00942: table or view does not exist

RESULT

Thus The VIEW has been created and updated successfully in RDBMS

EX: NO: 4.1 LIBRARY INFORMATION SYSTEM

AIM

To design a schema for Library Information and solve the following queries using the schema.

SCHEMA DESCRIPTION

Book (Book_id, Title, Publisher)

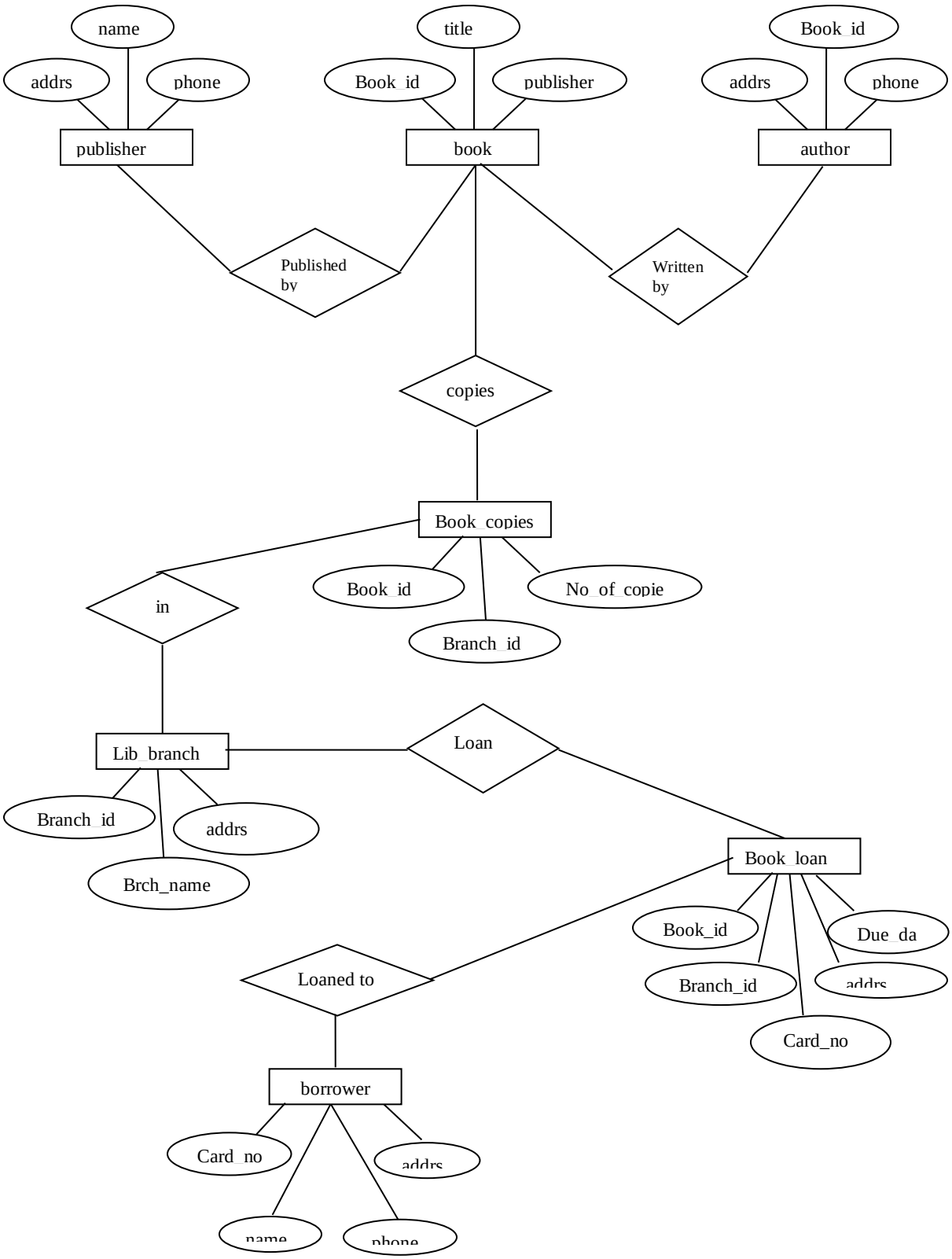
Publisher (Name, Addrs, Phone)

Book_copies (Book_id, Branch_id, No_of_copies)

Book_loan (Book_id, Branch_id, Card_no,Date_out, Due_date)

Lib_branch (Branch_id, Branch_name, Addrs)

Borrower (Card_no, Name, Addrs, Phone)


name
addrs
phone
publisher
title
Book_id
publisher
book
Book_id
addrs
phone
author
Published by
Written by
copies
Book_copies
Book_id
No_of_copie
Branch_id
in
Lib_branch
Loan
Branch_id
addrs
Brch_name
Book_loan
Book_id
Due_da
Branch_id
addrs
Card_no
Loaned to
borrower
Card_no
addrs
name
phone

CREATING TABLES

```
SQL> create table books
        (
        book_id varchar(5),
        title varchar(15),
        publisher varchar(20),
        primary key (book_id)
        );
```

Table created.

```
SQL> desc books;
```

Name	Null?	Type
BOOK_ID	NOT NULL	VARCHAR2(5)
TITLE		VARCHAR2(15)
PUBLISHER		VARCHAR2(20)

```
SQL> create table publisher
        (
        name varchar(20),
        addrs varchar(25),
        phone number(10)
        );
```

Table created.

```
SQL> desc publisher;
```

Name	Null?	Type
NAME		VARCHAR2(20)
ADDRS		VARCHAR2(25)
PHONE		NUMBER(10)

```
SQL> create table lib_branch
        (
        branch_id varchar(5),
        branch_name varchar(20),
        addrs varchar(25),
        primary key (branch_id)
        );
```

Table created.

SQL> desc lib_branch;

Name	Null?	Type
BRANCH_ID	NOT NULL	VARCHAR2(5)
BRANCH_NAME		VARCHAR2(20)
ADDRS		VARCHAR2(25)

```
SQL> create table book_author
        (
        book_id varchar(5),
        author varchar(15),
        addrs varchar(25),
        foreign key (book_id) references books(book_id)
        );
```

Table created.

SQL> desc book_author;

Name	Null?	Type
BOOK_ID		VARCHAR2(5)
AUTHOR		VARCHAR2(15)
ADDRS		VARCHAR2(25)

```
SQL> create table borrower
        (
        card_no varchar(5),
        name varchar(15),
        addrs varchar(25),
        phone number(10),
        primary key (card_no)
        );
```

Table created.

SQL> desc borrower;

Name	Null?	Type
CARD_NO	NOT NULL	VARCHAR2(5)
NAME		VARCHAR2(15)
ADDRS		VARCHAR2(25)
PHONE		NUMBER(10)

```
SQL> create table book_copies
        (
        book_id varchar(5),
        branch_id varchar(5),
        no_of_copies number(3),
        primary key (book_id,branch_id),
        foreign key (book_id) references books(book_id),
        foreign key (branch_id) references lib_branch(branch_id)
        );
```

Table created.

SQL> desc book_copies;

Name	Null?	Type
BOOK_ID	NOT NULL	VARCHAR2(5)
BRANCH_ID	NOT NULL	VARCHAR2(5)
NO_OF_COPIES		NUMBER(3)

```
SQL> create table book_loan
(
book_id varchar(5),
branch_id varchar(5),
card_no varchar(5),
date_out date,
due_date date,
foreign key (book_id) references books(book_id),
foreign key (branch_id) references lib_branch(branch_id),
foreign key (card_no) references borrower(card_no)
);
```

Table created.

SQL> desc book_loan;

Name	Null?	Type
BOOK_ID		VARCHAR2(5)
BRANCH_ID		VARCHAR2(5)
CARD_NO		VARCHAR2(5)
DATE_OUT		DATE
DUE_DATE		DATE

INSERTING VALUES INTO TABLES

//inserting values into table books

SQL> insert into books values('a501', 'cn', 'tata mac');
1 row created.

SQL> insert into books values('x555', 'dbms', 'tata mac');
1 row created.

SQL> insert into books values('c002', 'se', 'wrinklesons');
1 row created.

//displaying the values from table book

SQL> select * from books;

BOOK_ ID	TITLE	PUBLISHER
a501	cn	tata mac
x555	dbms	tata mac
c002	se	wrinklesons

//inserting values in the table publisher

SQL> insert into publisher values('tata mac','delhi',75820);
1 row created.

SQL> insert into publisher values('wrinklesons','kolkatta',44685);
1 row created.

//displaying the values in the table publisher

SQL> select * from publisher;

NAME	ADDRS	PHONE
tata mac	delhi	75820
wrinklesons	kolkatta	44685

//inserting the values in the table lib_branch

SQL> insert into lib_branch values('m101','senthil books','muthialpet');
1 row created.

SQL> insert into lib_branch values('l455','vinayaga book house','lawspet');
1 row created.

SQL> insert into lib_branch values('m100','pondy books','mg road');
1 row created

//displaying the values in the table lib_branch

SQL> select * from lib_branch;

BRANCH	BRANCH_NAME	ADDRS
m101	senthil books	muthialpet
l455	vinayaga book house	lawspet
m100	pondy books	mg road

//inserting values in the table book_author

SQL> insert into book_author values('a501','silverscrtz','banglore');
1 row created.

SQL> insert into book_author values('c002','balaji','pondy');
1 row created.

SQL> insert into book_author values('x555','suresh','lawspet');
1 row created.

//displaying the values from the table boo_author

SQL> select * from book_author;

BOOK_ ID	AUTHOR	ADDRS
a501	silverscrtz	banglore
c002	balaji	pondy
x555	suresh	lawspet

//inserting the values in the table borrower

SQL> insert into borrower values(101,'kumaran','23,mg st',22332);
1 row created.

SQL> insert into borrower values(155,'ajay','2nd main st',233232);
1 row created.

SQL> insert into borrower values(174,'saravanan','nehru st',252525);

1 row created.

//displaying the values in the table borrower

SQL> select * from borrower;

CARD_	NAME	ADDRS	PHONE
101	kumaran	23,mg st	22332
155	ajay	2nd main st	233232
174	saravanan	nehru st	252525

//inserting values in the table book_copies

SQL> insert into book_copies values('c002','m100',75);
1 row created.

SQL> insert into book_copies values('x555','l455',155);
1 row created.

SQL> insert into book_copies values('a501','m101',122);
1 row created.

//displaying the values in the table book_copies

SQL> select * from book_copies;

BOOK_ ID	BRANCH_ID	NO_OF_COPIES
c002	m100	75
x555	l455	155
a501	m101	122

//inserting the values in the table book_loan

```
SQL> insert into book_loan values('a501','l455',174,'12-feb-10','17-feb-10');
1 row created.

SQL> insert into book_loan values('x555','m101',155,'13-feb-10','18-feb-10');
1 row created.

SQL> insert into book_loan values('c002','m100',101,'18-feb-10','23-feb-10');
1 row created.

SQL> insert into book_loan values('x555','m100',101,'23-feb-10','28-feb-10');
1 row created.
```

//displaying the values in the table book_loan

SQL> select * from book_loan;

BOOK_ ID	BRANCH_ID	CARD_NO	DATE_OUT	DUE_DATE
a501	l455	174	12-FEB-10	17-FEB-10
x555	m101	155	13-FEB-10	18-FEB-10
c002	m100	101	18-FEB-10	23-FEB-10
x555	m100	101	23-FEB-10	28-FEB-10

RESULT

Thus the schema diagram for LIBRARY INFORMATION SYSTEM was studied and the queries are executed successfully.

EX: NO: 4.2 LOGISTIC MANAGEMENT SYSTEM

AIM

To design a schema for Logistic management and solve the following queries using the schema.

SCHEMA DESCRIPTION

Quotation(cusname , Goodsid, quan, destination, zipcode)

Shipper(shipperid, Shippername, shipperaddress, shippermobile, Goodsid)

Customer(customerid, customername, customeraddress, Goodsname, zipcode, Goodsid)

Trade(portid, Goodsid, ploading, pdispatch, TFM, originzip, deszip)

Stock(stockid, cname, quantity)

Container(stockid, containerid, contname, contsize, quant)

TABLE CREATION

//Creating table quotation

```
SQL> create table Quotation
  2  (
  3  cusname varchar2(15),
  4  Goodsid number primary key,
  5  quan number check(quan<=50 and quan>=1),
  6  destination varchar2(15),
  7  zipcode number check(zipcode<=30000 and zipcode>=200)
  8  );

Table created.
```

//Describing the table quotation

```
SQL> desc quotation;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 CUSNAME                                            VARCHAR2(15)
 GOODSID                                   NOT NULL NUMBER
 QUAN                                               NUMBER
 DESTINATION                                        VARCHAR2(15)
 ZIPCODE                                            NUMBER
```

//Creating table shipper

```
SQL> create table Shipper
  2  (
  3  shipperid number primary key,
  4  shippername varchar(15),
  5  shipperaddress varchar(15),
  6  shippermobile number(10),
  7  Goodsid number references Quotation(Goodsid)
  8  );

Table created.
```

//Describing the table shipper

```
SQL> desc shipper;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 SHIPPERID                                 NOT NULL NUMBER
 SHIPPERNAME                                        VARCHAR2(15)
 SHIPPERADDRESS                                     VARCHAR2(15)
 SHIPPERMOBILE                                      NUMBER(10)
 GOODSID                                            NUMBER
```

//Creating the table customer

```
SQL> create table customer
  2  (
  3  customerid number primary key,
  4  customername varchar2(15),
  5  customeraddress varchar2(15),
  6  Goodsname varchar(15),
  7  zipcode number check(zipcode<=30000 and zipcode>=200),
  8  Goodsid number references Quotation(Goodsid)
  9  );

Table created.
```

//Describing table customer

```
SQL> desc customer;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 CUSTOMERID                                NOT NULL NUMBER
 CUSTOMERNAME                                       VARCHAR2(15)
 CUSTOMERADDRESS                                    VARCHAR2(15)
 GOODSNAME                                          VARCHAR2(15)
 ZIPCODE                                            NUMBER
 GOODSID                                            NUMBER
```

//Creating table trade

```
SQL> create table trade
  2  (
  3  portid number primary key,
  4  Goodsid number references Quotation(Goodsid),
  5  ploading varchar2(15),
  6  pdispatch varchar(15),
  7  TFM varchar2(3),
  8  originzip number check(originzip<=30000 and originzip>=200),
  9  deszip number check(deszip<=30000 and deszip>=200)
 10  );

Table created.
```

//Describing the table trade

```
SQL> desc trade;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 PORTID                                    NOT NULL NUMBER
 GOODSID                                            NUMBER
 PLOADING                                           VARCHAR2(15)
 PDISPATCH                                          VARCHAR2(15)
 TFM                                                VARCHAR2(3)
 ORIGINZIP                                          NUMBER
 DESZIP                                             NUMBER
```

//Creating table stock

```
SQL> create table stock
  2  (
  3  stockid number primary key,
  4  cname varchar2(10),
  5  quantity number
  6  );

Table created.
```

//Describing the table stock

```
SQL> desc stock;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 STOCKID                                   NOT NULL NUMBER
 CNAME                                              VARCHAR2(10)
 QUANTITY                                           NUMBER
```

//Creating table container

```
SQL> create table container
  2  (
  3  stockid number references stock(stockid),
  4  containerid number,
  5  contname varchar2(15),
  6  contsize number check(contsize<=500  and contsize>=50),
  7  quant number
  8  );

Table created.
```

//Describing the table container

```
SQL> desc container;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 STOCKID                                            NUMBER
 CONTAINERID                                        NUMBER
 CONTNAME                                           VARCHAR2(15)
 CONTSIZE                                           NUMBER
 QUANT                                              NUMBER
```

//Inserting the values in table quotation

```
SQL> insert into Quotation values('deepika',0012,5,'mumbai',20136);
1 row created.
SQL> insert into Quotation values('revathy',0045,4,'konkan',10235);
1 row created.
SQL> insert into Quotation values('gowtham',0402,6,'kolkata',10236);
1 row created.
SQL> insert into Quotation values('karthik',0508,1,'pondy',22036);
1 row created.
SQL> insert into Quotation values('varun',0734,2,'kerala',10026);
1 row created.
SQL> insert into Quotation values('jeevith',9041,2,'pondy',22036);
1 row created.
```

//Inserting the values in table shipper

```
SQL> insert into shipper values(1015,'arnav', 'pondy', 9626806602, 0012);
1 row created.
SQL> insert into shipper values(1016,'sholk', 'chennai', 8015832696, 0045);
1 row created.
SQL> insert into shipper values(1017,'anant', 'manipur', 9994656160, 0402);
1 row created.
SQL> insert into shipper values(1018,'sidhu', 'gujarat', 979054797, 0508);
1 row created.
SQL> insert into shipper values(1019,'viren', 'konkan', 7418530950, 0734);
1 row created.
```

//Inserting values in table customer

```
SQL> insert into customer values(023, 'Karthik', 'pondy', 'electric_goods', 22036, 0508);

1 row created.

SQL> insert into customer values(025, 'varun', 'kerala', 'cereals', 10026, 0734);

1 row created.

SQL> insert into customer values(056, 'deepika', 'mumbai', 'fibre', 20136, 0012);

1 row created.

SQL> insert into customer values(043, 'gowtham', 'kolkata', 'spices', 10236, 0402);

1 row created.

SQL> insert into customer values(044, 'revathy', 'konkan', 'textile', 10235, 0045);

1 row created.
```

//Inserting values into table trade

```
SQL> insert into trade values(101, 0402, 'kerala', 'kolkata', 'PTP', 10026, 10236);

1 row created.

SQL> insert into trade values(102, 0012, 'konkan', 'mumbai', 'PTP', 10235, 20136);

1 row created.

SQL> insert into trade values(402, 0508, 'mumbai', 'pondy', 'DTP', 20136, 22036);

1 row created.

SQL> insert into trade values(202, 0734, 'kolkata', 'kerala', 'PTD', 10236, 10026);

1 row created.

SQL> insert into trade values(206, 0045, 'chennai', 'pondy', 'PTP', 20666, 22036);

1 row created.
```

//Inserting values into table stock

```
SQL> insert into stock values(23, 'fiber', 1400);

1 row created.

SQL> insert into stock values(25, 'mobile', 5000);

1 row created.

SQL> insert into stock values(45, 'textile', 8000);

1 row created.
```

//Displaying the values in table quotation

```
SQL> insert into container values(23, 10110, 'full', 420, 5);

1 row created.

SQL> insert into container values(45, 20111, 'full', 460, 4);

1 row created.
```

//Displaying the values in table quotation

```
SQL> select * from quotation;

CUSNAME            GOODSID       QUAN DESTINATION        ZIPCODE
--------------- ---------- ---------- --------------- ----------
deepika                 12          5 mumbai               20136
revathy                 45          4 konkan               10235
gowtham                402          6 kolkata              10236
karthik                508          1 pondy                22036
varun                  734          2 kerala               10026
jeevith               9041          2 pondy                22036
```

//Displaying the values in table shipper

```
SQL> select * from shipper;

 SHIPPERID SHIPPERNAME     SHIPPERADDRESS  SHIPPERMOBILE    GOODSID
---------- --------------- --------------- ------------- ----------
      1015 arnav           pondy              9626806602         12
      1016 sholk           chennai            8015832696         45
      1017 anant           manipur            9994656160        402
      1018 sidhu           gujarat             979054797        508
      1019 viren           konkan            7418530950        734
```

//Displaying the values in table customer

```
SQL> select * from customer;

CUSTOMERID CUSTOMERNAME    CUSTOMERADDRESS GOODSNAME          ZIPCODE    GOODSID
---------- --------------- --------------- --------------- ---------- ----------
        23 Karthik         pondy           electric_goods       22036        508
        25 varun           kerala          cereals              10026        734
        56 deepika         mumbai          fibre                20136         12
        43 gowtham         kolkata         spices               10236        402
        44 revathy         konkan          textile              10235         45
```

//Displaying the values in table trade

```
SQL> select * from trade;

    PORTID    GOODSID PLOADING        PDISPATCH       TFM  ORIGINZIP     DESZIP
---------- ---------- --------------- --------------- --- ---------- ----------
       101        402 kerala          kolkata         PTP      10026      10236
       102         12 konkan          mumbai          PTP      10235      20136
       402        508 mumbai          pondy           DTP      20136      22036
       202        734 kolkata         kerala          PTD      10236      10026
       206         45 chennai         pondy           PTP      20666      22036
```

//Displaying the values in table stock

```
SQL> select * from stock;

   STOCKID CNAME        QUANTITY
---------- ---------- ----------
        23 fiber            1400
        25 mobile           5000
        45 textile          8000
```

//Displaying the values in table container

```
SQL> select * from container;

   STOCKID CONTAINERID CONTNAME          CONTSIZE      QUANT
---------- ----------- --------------- ---------- ----------
        23       10110 full                   420          5
        45       20111 full                   460          4
```

MODIFICATION OF DATABASE

//Inserting the new record into the table quotation

```
SQL> insert into quotation values('astha', 8081, 5, 'konkon',20011);

1 row created.
```

```
SQL> select * from quotation;

CUSNAME            GOODSID       QUAN DESTINATION        ZIPCODE
--------------- ---------- ---------- --------------- ----------
deepika                 12          5 mumbai               20136
revathy                 45          4 konkan               10235
gowtham                402          6 kolkata              10236
karthik                508          1 pondy                22036
varun                  734          2 kerala               10026
jeevith               9041          2 pondy                22036
astha                 8081          5 konkon               20011

7 rows selected.
```

//Deleting a record from the table quotation

```
SQL> delete from quotation where goodsid=8081;

1 row deleted.

SQL> select * from quotation;
CUSNAME            GOODSID       QUAN DESTINATION        ZIPCODE
--------------- ---------- ---------- --------------- ----------
deepika                 12          5 mumbai               20136
revathy                 45          4 konkan               10235
gowtham                402          6 kolkata              10236
karthik                508          1 pondy                22036
varun                  734          2 kerala               10026
jeevith               9041          2 pondy                22036

6 rows selected.
```

//Inserting the new record into the table shipper

```
SQL> insert into shipper values(1013,'abhai', 'chennai', 9626806603, 9041);

1 row created.
SQL> select * from shipper;

 SHIPPERID SHIPPERNAME     SHIPPERADDRESS  SHIPPERMOBILE    GOODSID
---------- --------------- --------------- ------------- ----------
      1015 arnav           pondy              9626806602         12
      1016 sholk           chennai            8015832696         45
      1017 anant           manipur            9994656160        402
      1018 sidhu           gujarat             979054797        508
      1019 viren           konkan             7418530950        734
      1013 abhai           chennai            9626806603       9041

6 rows selected.
```

//Updating the shippername in the table shipper

```
SQL> update shipper set shippername='ganesh' where goodsid=9041;

1 row updated.
SQL> select * from shipper;

 SHIPPERID SHIPPERNAME     SHIPPERADDRESS  SHIPPERMOBILE    GOODSID
---------- --------------- --------------- ------------- ----------
      1015 arnav           pondy              9626806602         12
      1016 sholk           chennai            8015832696         45
      1017 anant           manipur            9994656160        402
      1018 sidhu           gujarat             979054797        508
      1019 viren           konkan             7418530950        734
      1013 ganesh          chennai            9626806603       9041

6 rows selected.
```

JOINS

//To compare each cusname from quotation and goodsid from customer using goodsid with right outer join.

```
SQL> select quotation.cusname, customer.goodsid
  2  from quotation
  3  right outer join customer
  4  on quotation.goodsid=customer.goodsid;

CUSNAME            GOODSID
--------------- ----------
deepika                 12
revathy                 45
gowtham                402
karthik                508
varun                  734
```

// To compare each cusname from quotation and goodsid from customer using goodsid with left outer join

```
SQL> select quotation.cusname, customer.goodsid
  2  from quotation
  3  left outer join customer
  4  on quotation.goodsid=customer.goodsid;

CUSNAME            GOODSID
--------------- ----------
karthik                508
varun                  734
deepika                 12
gowtham                402
revathy                 45
jeevith

6 rows selected.
```

AGGREGATE FUNCTION

//COUNT – To find the number tuples in the attribute TFM from the table trade

```
SQL> select count(TFM) from trade;

COUNT(TFM)
----------
         5
```

//AVG – To calculate the average value of attribute quantity from the table quotation

```
SQL> select avg(quan) from quotation;

 AVG(QUAN)
----------
3.33333333
```

//MAX – To compute the Maximum value of attribute goodsid from the table Trade

```
SQL> select max(goodsid) from trade;

MAX(GOODSID)
------------
         734
```

//MIN - To compute the Minimum value of attribute goodsid from the table Trade

```
SQL> select min(goodsid) from trade;

MIN(GOODSID)
------------
          12
```

//SUM – To compute the Total amount of the attribute quantity from the table quotation

```
SQL> select sum(quan) from quotation;

 SUM(QUAN)
----------
        20
```

NESTED SUBQUERIES

//Select the customer details from customer who belongs to destination of the customer 'jeevith'

```
SQL> select * from quotation where destination=
  2  (select destination from quotation where cusname='jeevith');

CUSNAME           GOODSID       QUAN DESTINATION        ZIPCODE
--------------- ---------- ---------- --------------- ----------
karthik               508          1 pondy                22036
jeevith              9041          2 pondy                22036
```

//To display the container details who belong to quant of the containerid =21002

```
SQL> select * from container where quant=
  2  (select quant from container where containerid=21002);

no rows selected
```

//To display the goodsid, cusname from the table quotation where goodsid not present in the table customer.

```
SQL> select goodsid, cusname from quotation where goodsid not in
  2  (select goodsid from customer);

   GOODSID CUSNAME
---------- ---------------
      9041 jeevith


SQL> select ploading, pdispatch, tfm from trade where deszip <
  2  all(19000, 23000);

PLOADING        PDISPATCH       TFM
--------------- --------------- ---
kerala          kolkata         PTP
kolkata         kerala          PTD
```

TRIGGER

```
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE TRIGGER cotrg AFTER INSERT ON container
  2  FOR EACH ROW
  3  DECLARE
  4  oldquan  NUMBER(10);
  5  quant  NUMBER(10);
  6  newquan  NUMBER(10);
  7  BEGIN
  8  SELECT quantity INTO oldquan FROM stock WHERE stockid=:new.stockid;
  9  quant:=:new.quant;
 10  newquan:=oldquan-quant;
 11  IF (quant<oldquan) THEN
 12  UPDATE stock SET quantity=newquan WHERE stockid=:new.stockid;
 13  ELSE
 14  RAISE_APPLICATION_ERROR(-20002,'Insufficient goods');
 15  END IF;
 16  END;
 17  /

Trigger created.
SQL> insert into container values(25, 20101, 'half', 410, 100);

1 row created.
```

```
SQL> select * from stock;

   STOCKID CNAME        QUANTITY
---------- ---------- ----------
        23 fiber            1400
        25 mobile           4900
        45 textile          8000

SQL> select * from container;

   STOCKID CONTAINERID CONTNAME          CONTSIZE      QUANT
---------- ----------- --------------- ---------- ----------
        23       10110 full                   420          5
        45       20111 full                   460          4
        25       20101 half                   410        100
```

RESULT

Thus the schema diagram for LOGISTIC MANAGEMENT SYSTEM was studied and the queries are executed successfully.

EX: NO: 4.3 Employee details using JDBC data base connectivity

AIM

To develop a Employee details for the any organization.

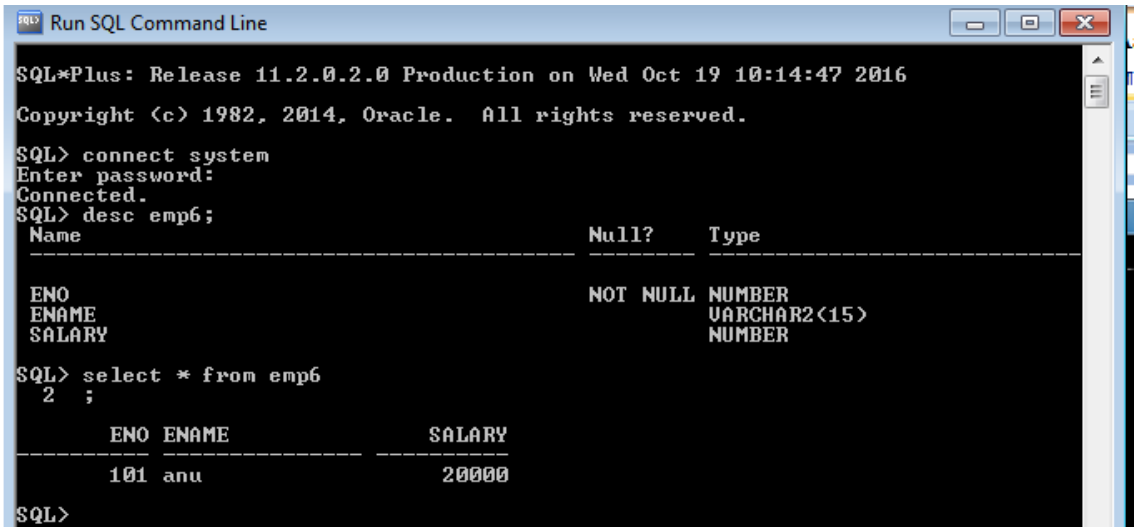
SCHEMA DESCRIPTION

STEP 1: Start
STEP 2: Create the Emp1 table with its essential attributes(Emp1(Eno, Ename,salary))

STEP 3: Insert attribute values into the emp1 table using jdbc connectivity
STEP 4: Update the attribute values into the emp1 table using jdbc connectivity
STEP 5: Delete the attribute values into the emp1 table using jdbc connectivity

STEP 6: Get the information of employee using select queries from the emp1 table using jdbc connectivity

STEP 5: Stop



Insert.java

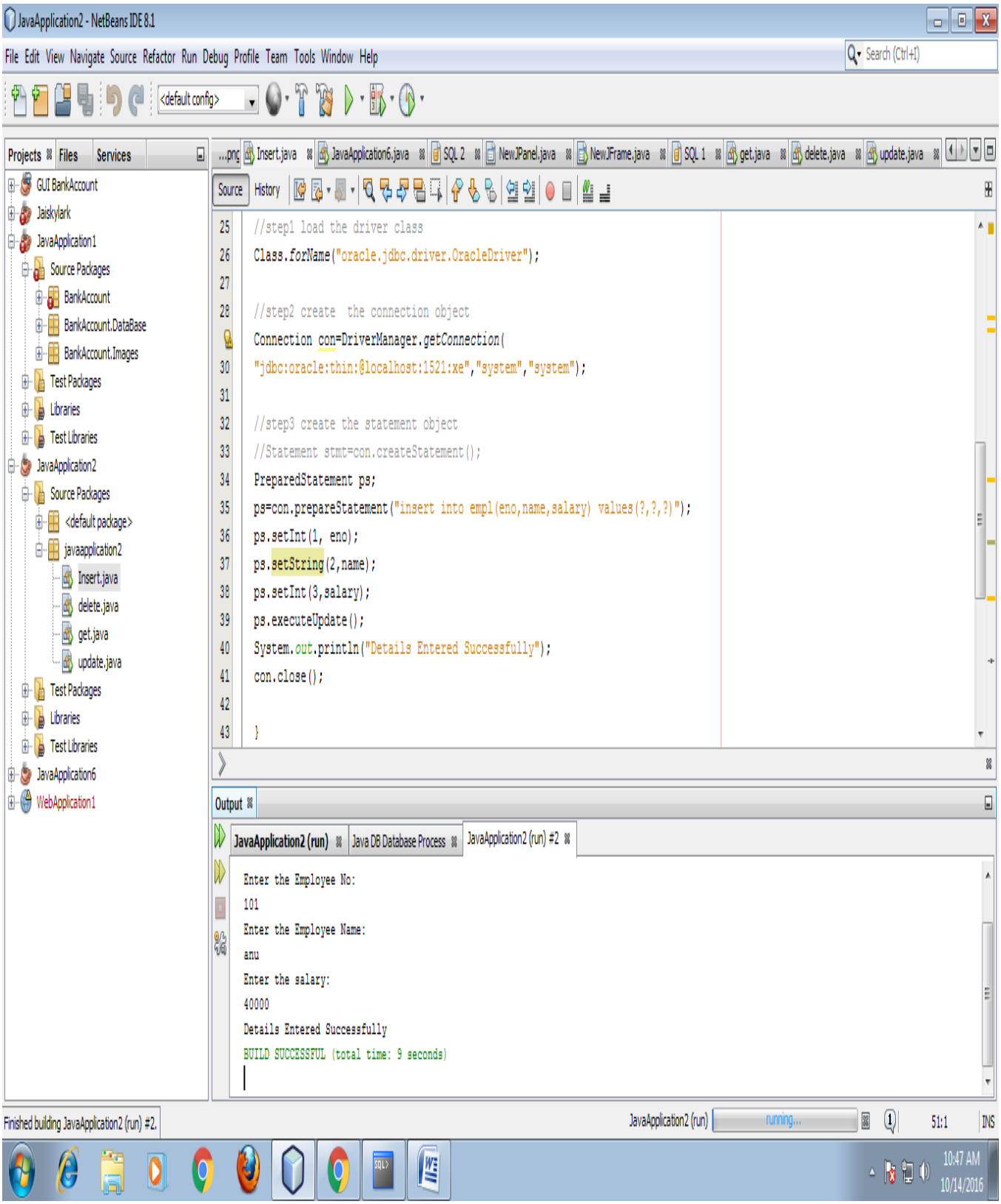
```
package javaapplication2;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.*;
import java.util.Scanner;
class Insert{
public static void main(String args[]){
try{
   Scanner s=new Scanner(System.in);
   System.out.println("Insert the Values in Table");
   System.out.println("Enter the Employee No:");
   int eno=s.nextInt();
   System.out.println("Enter the Employee Name:");
   String name=s.next();
   System.out.println("Enter the salary:");
   int salary=s.nextInt();
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","system");

//step3 create the statement object
//Statement stmt=con.createStatement();
PreparedStatement ps;
ps=con.prepareStatement("insert into empl(eno,name,salary) values(?,?,?)");
ps.setInt(1, eno);
ps.setString(2,name);
ps.setInt(3,salary);
ps.executeUpdate();
System.out.println("Details Entered Successfully");
con.close();

}
catch(Exception e)
{
   System.out.println(e);
}

}
}
```



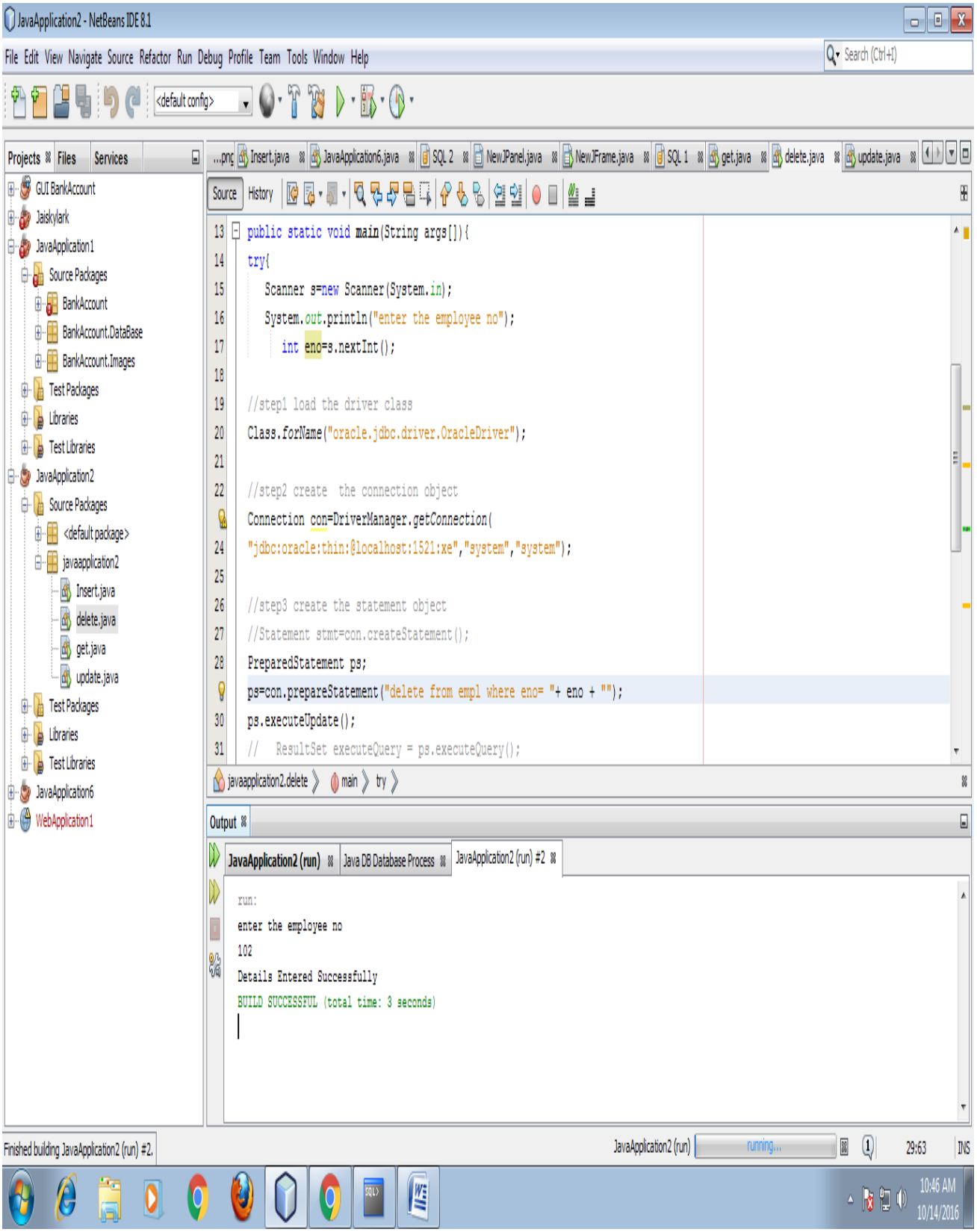
Delete.java

```
package javaapplication2;
import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.*;
import java.util.Scanner;
class delete{
public static void main(String args[]){
try{
   Scanner s=new Scanner(System.in);
   System.out.println("enter the employee no");
     int eno=s.nextInt();
    //step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","system");

//step3 create the statement object
//Statement stmt=con.createStatement();
PreparedStatement ps;
ps=con.prepareStatement("delete from empl where eno= "+ eno + "");
ps.executeUpdate();
System.out.println("Details Entered Successfully");
con.close();
}
catch(Exception e)
{
   System.out.println(e);
}
  }
}
```

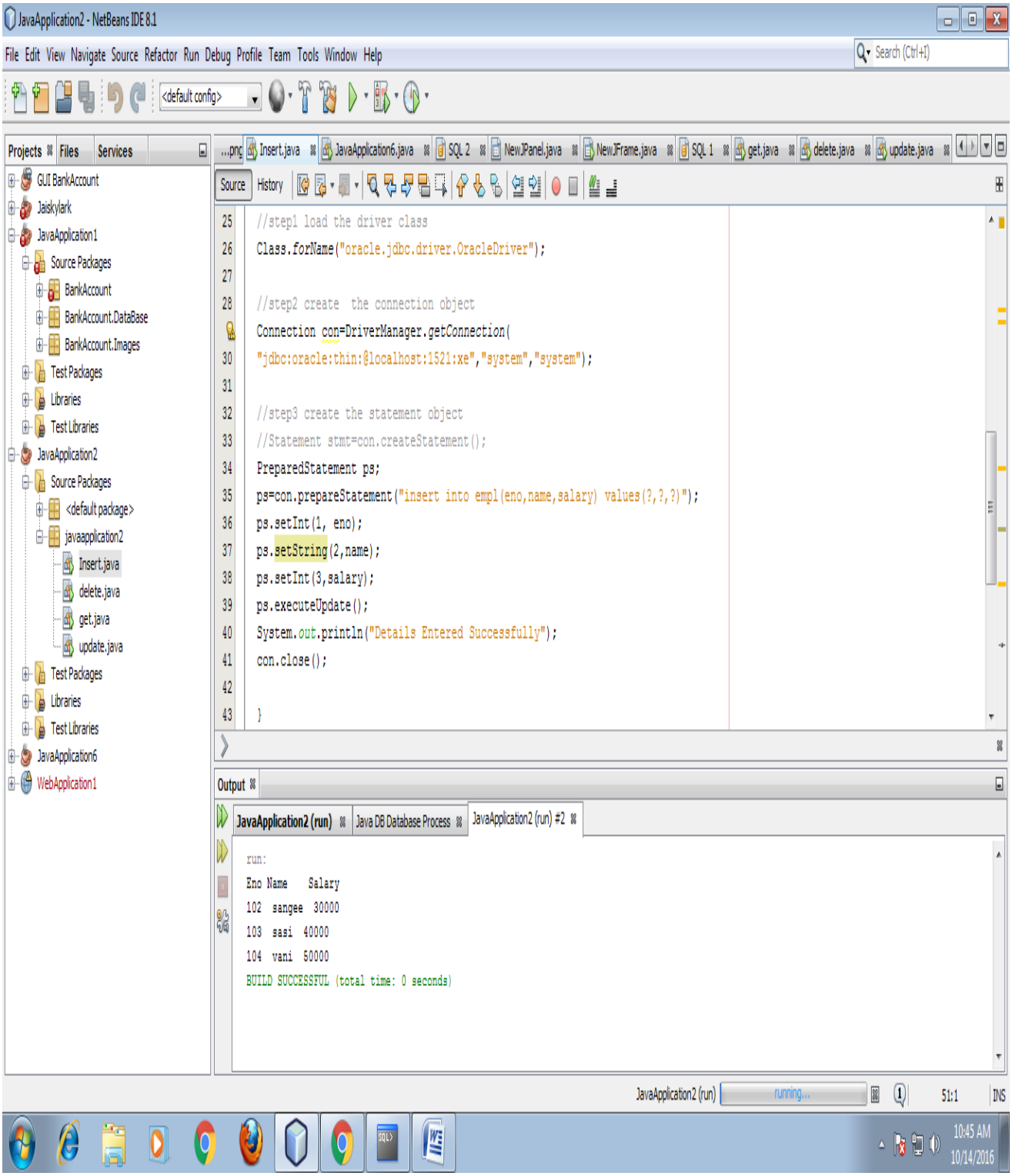

JavaApplication2 - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
GUI BankAccount
Jaiskylark
JavaApplication1
Source Packages
BankAccount
BankAccount.DataBase
BankAccount.Images
Test Packages
Libraries
Test Libraries
JavaApplication2
Source Packages
<default package>
javaapplication2
Insert.java
delete.java
get.java
update.java
Test Packages
Libraries
Test Libraries
JavaApplication6
WebApplication1
...png Insert.java JavaApplication6.java SQL 2 NewJPanel.java NewJFrame.java SQL 1 get.java delete.java update.java
Source History
13 public static void main(String args[]){
14 try{
15 Scanner s=new Scanner(System.in);
16 System.out.println("enter the employee no");
17 int eno=s.nextInt();
18
19 //step1 load the driver class
20 Class.forName("oracle.jdbc.driver.OracleDriver");
21
22 //step2 create the connection object
24 "jdbc:oracle:thin:@localhost:1521:xe","system","system");
25
26 //step3 create the statement object
27 //Statement stmt=con.createStatement();
28 PreparedStatement ps;
30 ps.executeUpdate();
31 // ResultSet executeQuery = ps.executeQuery();
Connection con=DriverManager.getConnection(
ps=con.prepareStatement("delete from empl where eno= "+ eno + "");
javaapplication2.delete main try
Output
JavaApplication2 (run) Java DB Database Process JavaApplication2 (run) #2
run:
enter the employee no
102
Details Entered Successfully
BUILD SUCCESSFUL (total time: 3 seconds)
Finished building JavaApplication2 (run) #2.
JavaApplication2 (run) running...
29:63 INS
10:46 AM
10/14/2016

Get.java

```
package javaapplication2;
import java.sql.*;
class get{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");
 //step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","system");
  //step3 create the statement object
Statement stmt=con.createStatement();
  //step4 execute query
ResultSet rs=stmt.executeQuery("select * from empl");
System.out.println("Eno Name    Salary");
while(rs.next())
System.out.println(+rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));
  // System.out.println(eno + "  " + name+"   "+salary);
//step5 close the connection object
con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```


JavaApplication2 - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects
Files
Services
GUI BankAccount
Jaiskylark
JavaApplication1
Source Packages
BankAccount
BankAccount.DataBase
BankAccount.Images
Test Packages
Libraries
Test Libraries
JavaApplication2
Source Packages
<default package>
javaapplication2
Insert.java
delete.java
get.java
update.java
Test Packages
Libraries
Test Libraries
JavaApplication6
WebApplication1
...png
Insert.java
JavaApplication6.java
SQL 2
NewJPanel.java
NewJFrame.java
SQL 1
get.java
delete.java
update.java
Source
History
25 //step1 load the driver class
26 Class.forName("oracle.jdbc.driver.OracleDriver");
27
28 //step2 create the connection object
Connection con=DriverManager.getConnection(
30 "jdbc:oracle:thin:@localhost:1521:xe","system","system");
31
32 //step3 create the statement object
33 //Statement stmt=con.createStatement();
34 PreparedStatement ps;
35 ps=con.prepareStatement("insert into empl(eno,name,salary) values(?,?,?)");
36 ps.setInt(1, eno);
37 ps.setString(2,name);
38 ps.setInt(3,salary);
39 ps.executeUpdate();
40 System.out.println("Details Entered Successfully");
41 con.close();
42
43 }
Output
JavaApplication2 (run)
Java DB Database Process
JavaApplication2 (run) #2
run:
Eno Name Salary
102 sangee 30000
103 sasi 40000
104 vani 50000
BUILD SUCCESSFUL (total time: 0 seconds)
JavaApplication2 (run)
running...
51:1
INS
10:45 AM
10/14/2016

Update.java

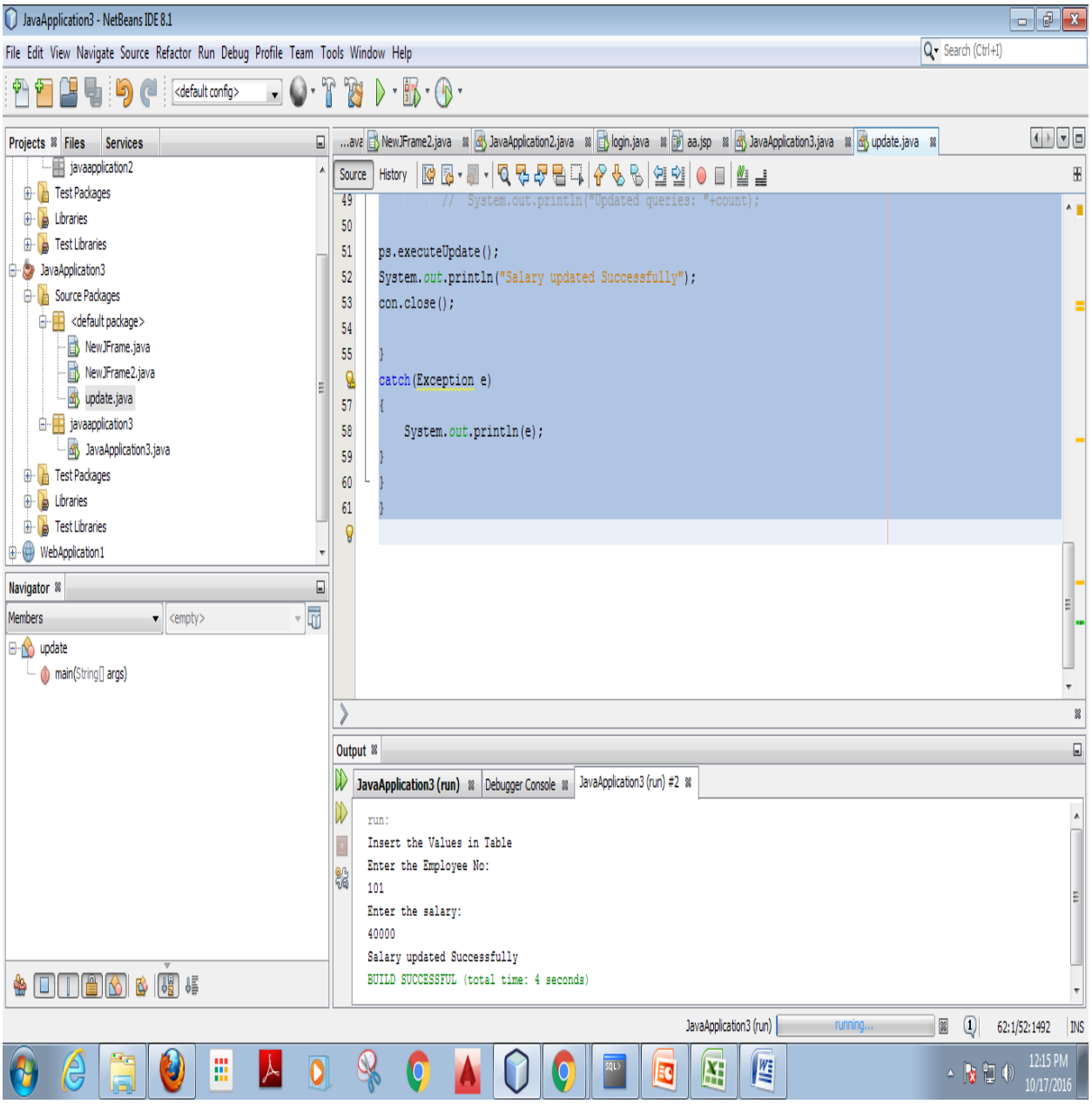
```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.*;
import java.util.Scanner;
public class update {

public static void main(String args[])
{
try{
    Scanner s=new Scanner(System.in);
   System.out.println("Insert the Values in Table");
    System.out.println("Enter the Employee No:");
    int eno=s.nextInt();
   System.out.println("Enter the salary:");
   int salary=s.nextInt();
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","it","it");

//step3 create the statement object
//Statement stmt=con.createStatement();
PreparedStatement ps;
ps=con.prepareStatement("update emp6 set salary=? where eno=?");
ps.setInt(1, salary);
ps.setInt(2,eno);
ps.executeUpdate();
System.out.println("Salary updated Successfully");
con.close();

}
catch(Exception e)
{
   System.out.println(e);
}
}
}
```



RESULT

Thus the schema diagram for Employee details was studied and the queries are executed successfully.