SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203.

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

LAB MANUAL

1922404 DATABASE MANAGEMANT SYSTEM & MINING LABORATORY

ACADEMIC YEAR 2022-2023

(EVEN SEMESTER)

FOURTH SEMESTER

Prepared By

Mr. R. Sankaranarayanan, Asst.Prof.(O.G)

1922404 DATABASE MANAGEMENT SYSTEMS & MINING LABORATORY L T P C
0 0 4 2

AIM:

The aim of this laboratory is to inculcate the abilities of applying the principles of the database management systems &mining. This course aims to prepare the students for projects where a proper implementation of databases will be required.

OBJECTIVES:

1. Learn to create and use a database
2. Be familiarized with a query language
3. Have hands on experience on DDL Commands
4. Have a good understanding of DML Commands and DCL commands
5. Be Exposed to basic algorithms of Data Mining

List of Experiments

1. Creation of a database and writing SQL queries to retrieve information from the database.
2. Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.
3. Creating an Employee database to set various constraints and Creation of Views Indexes, Save point.
4. Joins and Nested Queries.
5. Study of PL/SQL block.
6. Write a PL/SQL block to satisfy some conditions by accepting input from the user.
7. Write a PL/SQL block that handles all types of exceptions.
8. Creation of Procedures.
9. Creation of database triggers and functions
10. Creation of Database in Ms Access.
11. Database connectivity using Front End Tools (Application Development using Oracle/ Mysql)
12. Apriori Algorithm.
13. FP-Growth Algorithm.
14. Decision Tree Algorithm.

Mini Project
a) Inventory Control System.

b) Material Requirement Processing.

c) Hospital Management System.

d) Railway Reservation System.

e) Personal Information System

TOTAL: 60 PERIODS

LAB EQUIPMENT FOR A BATCH OF 30 STUDENTS:

SOFTWARE:

Systems with MySql, Visual Studio, Systems with Oracle 11g Client

HARDWARE:

Standalone Desktops: 30 Nos.

OUTCOMES:

At the end of the course, the student should be able to:

1. Use typical data definition and manipulation commands.

2. Design application to test nested and join queries.

3. Implement simple application that use views.

4. Implement application that requires front end tools.

5. Apply data mining techniques to large data sets

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

1. To afford the necessary background in the field of Artificial Intelligence and data Science to deal with engineering problems to excel as engineering professionals in industries.
2. To improve the qualities like creativity, leadership, teamwork and skill thus contributing towards the growth and development of society.
3. To develop ability among students towards innovation and entrepreneurship that caters to the needs of Industry and society.
4. To inculcate and attitude for life-long learning process through the use of Artificial Intelligence and Data Science sources.
5. To prepare then to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAM OUTCOMES (POs) ENGINEERING GRADUATES WILL BE ABLE TO:

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs):

1. Design and develop secured database applications with data analytical approaches of data preprocessing, optimization, visualization techniques and maintenance using state of the art methodologies based on ethical values.
2. Design and develop intelligent systems using computational principles, methods and systems for extracting knowledge from data to solve real time problems using advanced technologies and tools.
3. Design, plan and setting up the network that is helpful for contemporary business environments using latest software and hardware.
4. Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.

COURSE OUTCOMES:

Course Name:1922404&DBMS &MINING

Lab Year of Study:2021-2022

1922404.1	Use typical data definitions and manipulation commands.
1922404.2	Design applications to test Nested and Join Queries
1922404.3	Implement simple applications that use Views
1922404.4	Implement applications that require a Front-end Tool
1922404.5	Apply data mining techniques to large data sets

CO- PO MATRIX

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
1904002.1	3	2	-	-	-	-	-	-	-	-	-	-
1904002.2	2	2	2	-	2	-	-	-	-	-	-	-
1904002.3	3	2	2	-	2	-	-	-	-	-	-	-
1904002.4	3	2	2	-	2	-	-	-	-	-	-	-
1904002.5	-	2	2	-	-	-	-	-	-	-	-	-

JUSTIFICATION:

1904002.1	PO1	3	The knowledge can be enhanced byusing data definitions.
	PO2	2	Identify and analyze the complex engineering problems in data manipulation commands.
1904002.2	PO1	2	Applying the knowledge about Nested and Join Queries.
	PO2	2	Analyze and acquire the knowledge about the problem to be solved
	PO3	2	Design and develop the solution using nested and join queries.
	PO5	2	Appropriate techniques can be applied using queries.
1904002.3	PO1	3	The knowledge about the views to solve complex engineering problems
	PO2	2	Analyze, Understand and review various views to solve various computing problems
	PO3	2	The knowledge about various views can be applied to design efficient solutions to complex engineering problems
	PO5	2	Appropriate techniques can be applied using views for a given problem
1904002.4	PO1	3	To obtain basic knowledge of front end tool
	PO2	2	Analyze and acquire the knowledge about front end tool to be solved

	PO3	2	Design and develop the solution using front end tool
	PO5	2	Usage of modern technique that applies tool to solve complex problem.
1904002.5	PO2	2	Analyze the use of Tables, Views, Functions and Procedures to solve various computing problems

Program level course –PO matrix

C	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
1904002.	3	2	3	-	2	-	-	-	-	-	-	-

CO-PSO Matrix

	PSO 1	PSO 2	PSO3	PSO4
1904002.1	**3**	**2**	-	-
1904002.2	**3**	**3**	-	-
1904002.3	**3**	**2**	-	-
1904002.4	**3**	**2**	-	-
1904002.5	**3**	-	-	-

JUSTIFICATION:

1904002.1	PSO1	3	Develop the applications with data bases using data definitions and manipulation commands
	PSO2	2	Design and develop applications using data manipulation commands
1904002.2	PSO1	3	Students can design well planned database with the knowledge of .Nested and Join Queries

	PSO2	3	Design and develop solutions with risk assessment using Nested and Join Queries
1904002.3	PSO1	3	Develop the applications based on views
	PSO2	2	Can design and develop solutions for the complex database design.
1904002.4	PSO1	3	Developing application using Front-end Tool
	PSO2	2	Design solutions using Front-end Tool
1904002.5	PSO1	3	Can implement data mining techniques like Apriori algorithm,decision tree ,FP growth to large sets

CO	PSO1	PSO2	PSO3	PSO4
1904002.1	3	2	-	-

ASSESSMENT METHOD

MARK SPLIT UP	
AIM	**05**
PRE LAB VIVA QUESTIONS	**15**
OBSERVATION	**10**
QUERY/ PL SQL PROGRAM	**30**
OUTPUT	**20**
POST LAB VIVA QUESTIONS	**15**
RESULT	**05**
TOTAL	**100**

LIST OF EXPERIMENTS

1. Creation of a database and writing SQL queries to retrieve information from the database.

2. Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.

3. Creating an Employee database to set various constraints and Creation of Views Indexes, Save point.

4. Joins and Nested Queries.

5. Study of PL/SQL block.

6. Write a PL/SQL block to satisfy some conditions by accepting input from the user.

7. Write a PL/SQL block that handles all types of exceptions.

8. Creation of Procedures.

9. Creation of database triggers and functions

10.Creation of Database in Ms Access.

11.Database connectivity using Front End Tools (Application Development using Oracle/ Mysql)

Mini Project

a) Inventory Control System.

b) Material Requirement Processing.

c) Hospital Management System.

d) Railway Reservation System.

e) Personal Information System

Ex: No: 01 Creation of a database and writing SQL queries to retrieve information from the database.

AIM:

To create a database and writing SQL queries to retrieve information from the database.

DATA DEFINITION LANGUAGE

It is used to communicate with database. DDL is used to:

- Create an object
- Alter the structure of an object
- To drop the object created.

DDL COMMAND:

- CREATE
- ALTER
- DROP
- TRUNCATE
- COMMENT
- RENAME

CREATE TABLE

	Column 1	Column 2		Column n
Tuple or record 1 →				
Tuple or record 2 →				
. . .				
Tuple or record n →				

A table is uniquely identified by its name and consists of rows that contain the stored information, each row containing exactly one tuple (or record). A table can have one or more columns. A column is made up of a column name and a data type, and it describes an attribute of the tuples. The structure of a table, also called relation schema, thus is defined by its attributes. The type of information to be stored in a table is defined by the data types of the attributes at table creation time.

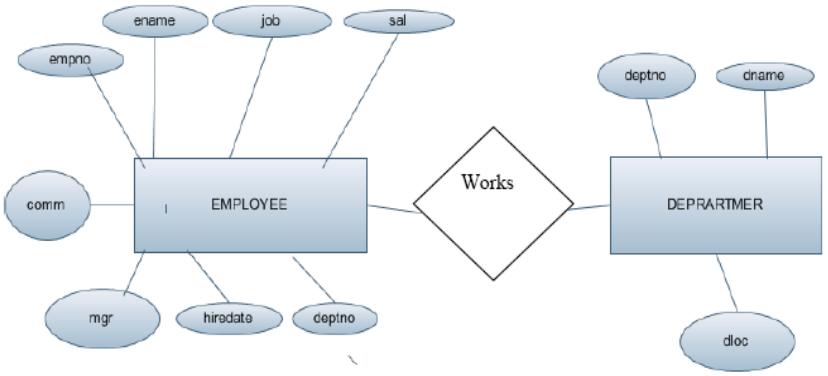
Oracle offers the following basic data types:

Sl.No	Data Type	Description
1	**Char(n)**	Character String . n is the size of variable. Maximum size is 255 characters. The default size is 1
2	**Varchar2(n)**	Character string . n is the size of the variable
3	**Number**	Defines Numeric data type with space for 40 digit and space for sign and

		decimal point
4	**Number (n)**	Numeric variable.n is the size of the variable
5	**Number(n,d)**	Numeric variable.n is the size of variable and d is the size if the decimal point

SQL uses the terms table, row, and column for relation, tuple, and attribute, respectively. A table can have up to 254 columns which may have different or same data types and sets of values (domains), respectively. Possible domains are alphanumeric data (strings), numbers and date formats.

Sample Databases used for illustration of SQL Commands is given below with ER Diagram and corresponding Relational Model with suitable data entered in the tables.



Relation between Employee and Department in Works is many –to-one .

DATABASE for EMPLOYEE and DEPARTMENT Entities

EMP Table given with sample Data

EMPNO	ENAME	JOB	MGR	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902 17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698 20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698 22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839 02-APR-81	2975		20
7654	MARTIN		7698 28-SEP-81	1250	1400	30

DEPT Table given with Sample Data

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

QUERY: 01

Q1: Write a query to create a table employee with empno, ename, designation, and salary.

Syntax: It is used to create a table

SQL: CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE),

COLUMN NAME.2 <DATATYPE> (SIZE) ……);

Constraints with Table Creation:

Constraints are condition for the data item to be stored into a database. There are two types of Constraints viz., Column Constraints and Table Constraints.

Syntax:

[CONSTRAINT constraint name] {[NOT] NULL / UNIQUE / PRIMARY KEY}(Column[,column]..) FOREIGN KEY (column [, colum]…)

REFERENCES table [ON DELETE CASCADE]

[CHECK (condition)]

TABLE DESCRIPTION

It is used to view the table structure to confirm whether the table was created correctly.

QUERY: 02

Q2: Write a query to display the column name and data type of the table employee.

Syntax: This is used to view the structure of the table.

SQL: DESC <TABLE NAME>;

QUERY: 03

Q3: Write a query for create a table from an existing table with all the fields

Syntax: syntax for create a table from an existing table with all fields.

SQL> CREATE TABLE <TRAGET TABLE NAME> SELECT * FROM<SOURCE TABLE NAME>;

QUERY: 04

Q4: Write a query for create a table from an existing table with selected fields

Syntax: Syntax for create a table from an existing table with selected fields.

SQL> CREATE TABLE <TRAGET TABLE NAME> AS SELECT EMPNO, ENAMEFROM <SOURCE TABLE NAME>;

DRL-DATA RETRIEVAL IMPLEMENTING ON SELECT COMMANDS

Command:

SQL> CREATE TABLE EMP(

EMPNO	NUMBER (4),
ENAME	VARCHAR2 (10),
JOB	VARCHAR2(20),
MGR	NUMBER(4),
HIREDATE	DATE,

```
        SAL         NUMBER(8,2),
        DEPTNO      NUMBER(3)
    );
```

Table created.

THE SELECT STATEMENT SYNTAX WITH ADDITIONAL CLAUSES:

```
Select  [ Distinct / Unique ] ( *columnname [ As alias}, ….]
From tablename
[ where  condition ]
[ Group BY group _by_expression ]
[Having  group_condition ]
[ORDER BY {col(s)|expr|numeric_pos} [ASC|DESC] [NULLS FIRST|LAST]];
```

BY USING BETWEEN / NOT / IN / NULL / LIKE

BETWEEN Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

IN Syntax

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

LIKE Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;
```

RESULT:

Thus to create a database and writing SQL queries to retrieve information from the database is verified successfully

Ex: No: 02 Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.

Aim : To performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.

DML (DATA MANIPULATION LANGUAGE)

- SELECT
- INSERT
- DELETE
- UPDATE

Syntax:
Insert:

insert into <tablename> values(val 1,val 2,...,val n);
insert into <tablename> values(&col 1,&col 2,...,&col n);

Select:

select <column name> from <tablename> where <condition>;

Update:

update <tablename> set <columnname>=value where <condition>;

INSERT

1. Insert the following values in emp table

Eno	Ename	Jdate	Job	Comm.	Deptno	Salary
7369	Smith	17-dec-80	Clerk	20	800	
7499	Allan	20-feb-81	Salesman	300	30	1600
7521	Ward	22-feb-81	Salesman	500	30	1250
7566	Jones	02-apr-81	Manager		20	2975
7654	Martin	28-sep-81	Salesman	1400	30	1250
7698	Blake	01-may-81	Manager		30	2850
7782	Clark	09-jun-81	Manger	10	2450	
7788	Scott	19-apr-87	Analyst	20	3000	
7839	King	17-nov-81	President		10	5000
7844	Turner	08-sep-81	Salesman		30	1500
7876	Adams	28-may-87	Clerk	20	1100	
7900	James	03-dec-81	Clerk	30	950	
7902	Ford	03-dec-81	Analyst	20	3000	
7934	Miller	28-jan-81	Clerk	10	1300	

2. Update salary of each employee by 200

3. Update salary of eno 7566 to 3000

4. Insert one empty row

5. Delete the employee with ename='Adams'

6. Select all values from the table emp

7. Select eno,ename of employee of salary>3000

8. Select all employee whose salary is >2000 and <3000

9. Select all employees except managers

10. Select list of all employees whose name starts with 'J'

11. Select list of all employees whose name is of length 4

12. Select eno, ename, salary of all employees working in deptno 10 & 20

13. Select list of employees whose 3rd letter of ename is is 'r'

14. Select list of employees who don't get commision

15. Select eno, ename, salary of all employee by increasing salary by 200 while displaying

16. Display eno of all manager of each dept in following format

17. Save the table emp with suitable name

18. In the newly created table

19. Save all changes made to table

ALTER & MODIFICATION ON TABLE

To modify structure of an already existing table to add one more columns and also modify the existing columns.

Alter command is used to:

1. Add a new column.
2. Modify the existing column definition.
3. To include or drop integrity constraint.

QUERY: 05

Q6: Write a Query to Alter the column EMPNO NUMBER (4) TO EMPNO NUMBER (6).

Syntax: The syntax for alter & modify on a single column.

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME><DATATYPE>(SIZE);

QUERY: 06
Q8. Write a query to add a new column in to employee

Syntax: To add a new column.
SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME><DATATYPE><SIZE>);

RESULT:

Thus the performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions. has been verified and executed successfully

Ex: No: 03 Creating an Employee database to set various constraints and Creation of Views Indexes, Save point.

AIM:

To Creating an Employee database to set various constraints and Creation of Views Indexes, Save point..

CONSTRAINTS

The definition of a table may include the specification of integrity constraints. Basically two types of constraints are provided: column constraints are associated with a single column whereas table constraints are typically associated with more than one column.

Primary Key | Unique | check | not null:

create table <tablename>(col1 datatype constraint <constraintname> [primary key|unique|check <colname with condition>|not null],…);

Foreign Key:

alter table <tablename> add constraint <constraintname> foreign key(<childtable colname>) references <parenttablename(cols)>;

Drop constraint:

alter table <tablename> drop constraint <constraintname>;

PRIMARY KEY CONSTRAINTS

A primary key avoids duplication of rows and does not allow null values. It can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never be changed and should never be null. A table should have only one primary key. If a primary key constraint is assigned to more than one column or combination of column is said to be composite primary key, which can contain 16 columns.

Column level constraints using primary key:

Syntax: Column level constraints using primary key.
SQL> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE)<TYPE OF CONSTRAINTS>, COLUMN NAME.1 <DATATYPE> (SIZE) ……);

Column level constraints using primary key with naming convention:

Syntax: syntax for column level constraints using primary key.

SQL >CREATE <OBJ.TYPE><OBJ.NAME> (
COL NAME.1 <DATATYPE> (SIZE)CONSTRAINTS <NAME OF CONSTRAINTS><TYPE OF CONSTRAINTS>,

COL NAME.2 <DATATYPE> (SIZE)…….……);

Table level primary key constraints:

Syntax: The syntax for table level constraints using primary key

SQL: >CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE),

COLUMN NAME.1 <DATATYPE> (SIZE),

CONSTRAINTS <NAME OF THE CONSTRAINTS><TYPE OF THE CONSTRAINTS>);

Table level constraint with alter command (primary key):

Syntax: The syntax for column level constraints using primary key.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),

COLUMN NAME.1 <DATATYPE> (SIZE));

[OR]

SQL> ALTER TABLE <TABLE NAME> ADD CONSTRAINTS <NAME OF THECONSTRAINTS>

<TYPE OF THE CONSTRAINTS><COLUMN NAME>);

REFERENCE /FOREIGN KEY CONSTRAINT

It enforces relationship between tables. To establish parent-child relationship between 2 tables having a common column definition, we make use of this constraint. To implement this, we should define the column in the parent table as primary key and same column in the child table as foreign key referring to the corresponding parent entry.

Foreign key

A column or combination of column included in the definition of referential integrity, which would refer to a referenced key.

Referenced key

It is a unique or primary key upon which is defined on a column belonging to the parent table.

Column level foreign key constraint

Syntax: Syntax for Column level constraints Using Primary key

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (
COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF CONSTRAINTS> ,
COLUMN NAME.1 <DATATYPE> (SIZE)……);

Column level foreign key constraint with naming conversions

Syntax: The syntax for column level constraints using primary key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF

CONSTRAINTS>,COLUMN NAME.1 <DATATYPE> (SIZE)…);

Child Table:

Syntax: syntax for column level constraints using foreign key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE) ,

COLUMN NAME2 <DATATYPE> (SIZE) **CONSTRAINT <CONST.NAME>**REFERENCES <TABLE NAME> (COLUMN NAME>…);

Table level foreign key constraints:

Parent Table:

Syntax:

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE) <TYPE OF CONSTRAINTS>, COLUMN NAME.1 <DATATYPE> (SIZE)…);

Child Table:

Syntax: The syntax for table level constraints using foreign key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),
COLUMN NAME2 <DATATYPE> (SIZE),
CONSTRAINT <CONST.NAME>REFERENCES <TABLE NAME> (COLUMN NAME>);

Table level foreign key constraints with alter command:

Parent Table:

Syntax:

SQL :>CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF CONSTRAINTS> , COLUMN NAME.1 <DATATYPE> (SIZE) …..……);

Child Table:

Syntax: The syntax for table level constraints using foreign key.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),
COLUMN NAME2 <DATATYPE> (SIZE));

Syntax:

SQL> ALTER TABLE <TABLE NAME> ADD CONSTRAINT <CONST. NAME>REFERENCES <TABLE NAME> (COLUMN NAME>);

CHECK CONSTRAINT

Check constraint can be defined to allow only a particular range of values .when the manipulation violates this constraint, the record will be rejected. Check condition cannot contain sub queries.

Column level checks constraint:

Syntax: syntax for column level constraints using check.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE)

CONSTRAINT <CONSTRAINTS NAME><TYPE OF CONSTRAINTS>(CONSTRAITNS CRITERIA) , COLUMN NAME2 <DATATYPE> (SIZE));

Check Constraint with Alter Command:

Syntax: Syntax for Table level constraints using Check.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),
(COLUMN NAME2 <DATATYPE> (SIZE),
CONSTRAINT<CONSTRAINTS NAME><TYPE OF CONSTRAINTS> (CONSTRAITNSCRITERIA)) ;

UNIQUE CONSTRAINT

It is used to ensure that information in the column for each record is unique, as with telephone or drivers license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value.

If unique key constraint is defined in more than one column i.e., combination of column cannot be specified. Maximum combination of columns that a composite unique key can contain is 16.

Column Level Constraint:
Syntax: syntax for column level constraints with unique.

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (<COLUMN NAME.1> <DATATYPE> (SIZE)
CONSTRAINT <NAME OF CONSTRAINTS><CONSTRAINT TYPE>,
(COLUMN NAME2 <DATATYPE> (SIZE));

NOT NULL CONSTRAINTS

While creating tables, by default the rows can have null value .the enforcement of not null constraint in a table ensure that the table contains values.

Column Level Constraint:

Syntax: syntax for column level constraints with not null

SQL :> CREATE <OBJ.TYPE><OBJ.NAME>(<COLUMN NAME.1><DATATYPE> (SIZE)
CONSTRAINT <NAME OF CONSTRAINTS> <CONSTRAINT TYPE>,
(COLUMN NAME2 <DATATYPE> (SIZE)) ;

NULL CONSTRAINTS

Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful.

- A null value is not equivalent to a value of zero.
- A null value will always evaluate to null in any expression.
- When a column name is defined as not null, that column becomes a mandatory

i.e., the user has to enter data into it.

- Not null Integrity constraint cannot be defined using the alter table command when the table contain rows.

Column Level Constraint:

Syntax: syntax for column level constraints with null

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (

<COLUMN NAME.1><DATATYPE> (SIZE) **CONSTRAINT** <NAME OF CONSTRAINTS> <CONSTRAINT TYPE>,(COLUMN NAME2 <DATATYPE> (SIZE)) ;

An implicit savepoint is marked before executing an **INSERT, UPDATE, or DELETE** statement. If the statement fails, a rollback to the implicit savepoint is done. Normally, just the failed SQL statement is rolled back, not the whole transaction; if the statement raises an unhandled exception, the host environment

Syntax:

SAVEPOINT<SAVEPOINT_NAME>;

Ex:

```
SQL> create table ORDER_PROCESSING(
                                    Order_ID number(3),
                                    Product_ID varchar2(10),
                                    Quantity number(3,2),
                                    Price number(4,2)
                                    );
```

Table created.

```
SQL> insert into ORDER_PROCESSING values(101,'RICE-22','6.5','30.50');
```

1 row created.

```
SQL> insert into ORDER_PROCESSING values(102,'OIL','2.0','90.50');
```

1 row created.

```
SQL> SELECT * FROM ORDER_PROCESSING;
ORDER_ID    PRODUCT_ID          QUANTITY          PRICE
----------  ----------          ----------        ----------
101         RICE-22       6.5           30.5
102         OIL                 2                 90.5
SQL> COMMIT;
```

Commit complete.

```
SQL> insert into ORDER_PROCESSING values(103,'BAGS','2','95');
```

1 row created.

```
SQL> insert into ORDER_PROCESSING values(104,'WATER BOTS','2','20');
```

1 row created.

```
SQL> SAVEPOINT A;
```

Savepoint created.

```
SQL> insert into ORDER_PROCESSING values(105,'EGG','8','40.50');
```

1 row created.

```
SQL> insert into ORDER_PROCESSING values(106,'SHAMPOO','1','75.50');
```

1 row created.

SQL> **SAVEPOINT B;**

Savepoint created.

SQL> insert into ORDER_PROCESSING values(107,'BAR SOAP','1','45.50');

1 row created.

SQL> insert into ORDER_PROCESSING values(108,'TONER','1','75.50');

1 row created.

SQL> **SAVEPOINT C;**

Savepoint created.

SQL> insert into ORDER_PROCESSING values(109,'SUGAR','2.0','60.50');

1 row created.

RESULT:

Thus the SQL commands has been verified and executed successfully for creating an Employee database to set various constraints and Creation of Views Indexes, Save point..

Ex: No: 04 Joins and Nested Queries.

AIM:

To execute and verify the SQL commands for various join operation.

JOINS:

Joins are used to retrieve the data from multiple tables.

Types of Joins:

1. EQUI_JOIN
2. NON EQUI_JOIN
3. SELF JOIN
4. OUTER JOIN
    4.1 Right outer join
    4.2 Left outer join
    4.3 Full outer join

1. **EQUI_JOIN**:

When tables are joined basing on a common column it is called EQUI_JOIN.

Ex:

select empno, ename, dname from emp, dept where emp.deptno = dept.deptno;

EMPNO	ENAME	DNAME
7369	SMITH	RESEARCH
7499	ALLEN	SALES
7521	WARD	SALES

Note:

We need to mention join conditions in the where clause.

In EQUI_JOINS we along use to equal to operator in join condition.

Ex:

SQL>Selete empno, ename, sal, job, dname, loc

from emp, dept

where emp.deptno = dept.deptno;

SQL>Selete empno, ename, sal, deptno, dname, loc

from emp, dept

where emp.deptno = dept.deptno;// error

SQL>Selete empno, ename, sal, emp.deptno, dname, loc

from emp, dept

where emp.deptno = dept.deptno; //valid

Note:

- we need to mention table name dot column(emp.deptno) name for the common column to resolve the any table.
- The common column can be retrieved from any of the table.
- We can filter the data from the result of join.

Ex:

SQL>Select empno, ename, sal, emp.deptno, dname, loc

from emp, dept

where emp.deptno = dept.deptno AND sal > 2000;

To improve the performance of the join we need mention table name dot column name for all the columns.

Ex:

SQL>Select emp.empno, emp.ename, emp.sal,emp.deptno, dept.dname, dept.loc

from emp,dept

where emp.deptno = dept.deptno AND sal > 2000;

Table alias:

- Table alias is an alternate name given to a table.
- By using a table alias length of the table reduces and at the same time performance is maintains.
- Table alias are create in same clause can be used in select clause as well as where clause.
- Table alias is temporary once the query is executed the table alias are losed.

Ex:

SQL>Select E.Empno, E.Ename, E.sal, E.deptno, D.Dname, D.loc from emp E, Dept D

where E.deptno = D.deptno;

Join the multiple tables(3 tables):

Select * from Areas;

City	**State**
Newyork	AP
Dallas	Mh

Ex:

SQL>Select E.empno, E.ename, E.sal,D.dname,A.state from emp E, dept D, Areas A

where E.deptno = D.deptno AND D.loc = A.city;

Note: To join 'n' tables we need n-1 conditions.

NON EQUI JOIN:

When we do not use NON EQUI JOIN to operator in the join condition is NON EQUI JOIN.

Ex:

SQL>Select * from SALGRADE;

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

SQL>Select e.empno, e.ename, e.sal, s.grade from emp e, salgrade s

where e.sal BETWEEN s.losal AND hisal;

EMPNO	ENAME	GRADE
7369	SMITH	1
7876	ADAMS	1
7900	JAMES	2

SQL>Select e.empno, e.ename, s.grade from emp e, salgrade s

where e.sal BETWEEN s.losal AND s.hisal AND s.grade = 4;

SELF JOIN:

When a table is joining to it self it is called self join. In self joins we need to create two table aliases for the same table.

SQL>Select empno, ename, job, mgr, from emp;

SQL>Select e.empno, e.ename, e.job, m.ename from emp e, emp m where e.mgr = m.empno;

Empno	**Ename Job**	**Ename**
7902	FORD ANALYST	JONES
7869	SCOTT CLERK	JONES
7900	JAMES SALESMAN	BLAKE

CARTESIAN PRODUCT:

When tables are joined without any join condition it is called Cartesian product. In the result we get all possible combination.

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e, dept d; //14*4=56 rows are selected

ANSI JOINS:

They are the three types.

INNER JOINS:

It is same as Equi join.

Ex:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e INNER JOIN dept d ON(e.deptno = d.deptno);

2.**NATURAL JOIN**:

It is same as Equi join.

Ex:

SQL>Select empno, ename, sal, deptno, dname,loc from NATURAL JOIN dept;

CROSS PRODUCT/CROSS JOIN:

It is same as Cartesian product.

Ex:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e CROSS JOIN dept d; //14*4 = 56 rows are displayed.

DEFAULT:

Ex:

SQL>Create table stu1(sno number(3), Sname varchar2(10), Marks number(3) default 100,

Doj Date DEFAULT sysdate);

SQL>Insert into stu1(sno, sname) values(101,'malli');

SQL>Insert into stu1 values(102,'ARUN',40,'11-JAN-09');

SQL>Insert into stu1 values (103,'KIRAN',NULL,'12-FEB-10');

SNO	SNAME	MARKS	DOJ
101	malli	100	26-JUN-12

102	ARUN	40	11-JAN-09
103	KIRAN		12-FEB-10

SUPER KEY:

Combination of columns which can be used unique key identify every row is called as super key.

OUTER JOINS:

It is extension of EQUI JOINS.

In outer joins we get match as well as non-matching rows.

(+) This called as outer join operator.

1. RIGHT OUTER JOIN:

SQL Syntax:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from emp e, dept d

where e.deptno(+) = d.deptno; //14 + 1 = 15 rows

empno	**ename**	**sal**	**deptno**	**dname**	**loc**
7900	james	950	30	sales	chicago
8963	adams	1400	20	clerk	newyork
6798	adams	2000	10	sales	india

ANSI SYNTAX OF RIGHT OUTER JOIN:

ANSI SYSTAX:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e RIGHT OUTER JOIN dept d ON(e.deptno = d.deptno);

LEFT OUTER JOIN:

SQL Syntax:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from emp e, dept d

where e.deptno = d.deptno(+); //14+3 = 17 row displayed

ANSI SYNTAX OF LEFT OUTER JOIN:

ANSI SYNTAX:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e LEFT OUTER JOIN dept d ON(e.deptno = d.deptno);

FULL OUTER JOIN:

ANSI SYNTAX:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e FULL OUTER JOIN dept d ON(e.deptno = d.deptno);

RESULT:

Thus the SQL commands has been verified and executed successfully for various join operations.

Ex: No: 05 **Study of PL/SQL block.**

AIM:

To write a PL/SQL block using different control (if, if else, for loop, while loop,…) statements.

PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. PL/SQL allows you to mix SQL statements with procedural statements like IF statement, Looping structures etc.

It is extension of SQL the following or advantages of PL/SQL.

1. We can use programming features like if statement loops etc.
2. PL/SQL helps in reducing network traffic.
3. We can have user defined error massages by using concept of exception handling.
4. We can perform related actions by using concept of Triggers.
5. We can save the source code permanently for repeated execution.

PL/SQL Block:

A PL/SQL programs called as PL/SQL block.

PL/SQL Block:

DECLARE

← Declaration of variable
Declaration of cursor ----------(OPTIONAL)
Declaration of exception

BEGIN

← Executable commands --------- (MANDATORY)

EXCEPTION

← Exception handlers ----------(OPTIONAL)

END;

← To execute the program / command

Declare:

This section is used to declare local variables, cursors, Exceptions and etc. This section is optional.

Executable Section:

This section contains lines of code which is used to complete table. It is mandatory.

Exception Section:

This section contains lines of code which will be executed only when exception is raised. This section is optional.

Simplest PL/SQL Block:

```
Begin
--------
--------
--------
END;
```

SERVEROUTPUT

This will be used to display the output of the PL/SQL programs. By default this will be off.

Syntax:

Set serveroutput on | off

Ex:

SQL> set serveroutput on

PL/SQL CONTROL STRUCTURES

PL/SQL has a variety of control structures that allow you to control the behaviour of the block as it runs. These structures include conditional statements and loops.

- If-then-else
- Case
  - Case with no else
  - Labeled case
  - Searched case
- Simple loop
- While loop
- For loop
- Goto and Labels

Program to find factorial of a number:

```
declare
        +.put_line(num1);
end;
```

Output:

```
Enter value for number: 5
old   7:  num:=&number;
new   7:  num:=5;
120
```

Program to reverse a number:

```
declare
        num number(10);
        num1 number(10);
        a number(2);
begin
        num1:=0;
        num:=&value;
        while num > 0 loop
                a := mod(num,10);
                num1 := (num1*10)+a;
                num := floor(num/10);
        end loop;
        dbms_output.put_line(num1);
 end;
```

Output:

```
Enter value for value: 789
old   7:  num:=&value;
new   7:  num:=789;
987
```

Program to generate Fibonacci series:

```
declare
        a number(6); b
        number(6); fib
        number(6); n
        number(6);
begin
        a:=-1;
        b:=1;
        n:=&n;
        while (n>0) loop
                fib:=a+b;
                a:=b;
                b:=fib;
                n:=n-1;
                dbms_output.put_line(fib);
        end loop;
end;
```

Output:

```
Enter value for n: 8
old  11: n:=&n;
new  11: n:=8;
0
1
1
2


3
5
8
13
```

.

RESULT:

Thus the Study of PL/SQL block has been implemented by various control structure are verified and executed successfully.

Ex: No: 06 WRITE A PL/SQL BLOCK TO SATISFY SOME CONDITIONS BY ACCEPTING INPUT FROM THE USER.

AIM:

To implement the PL/SQL block to satisfy some conditions by accepting input from the user.

Q1: Write PL/SQL block which will calculate some of two numbers and display the output?

```
DECLARE
        A number(2);
        B number(2);
        C number(3);
BEGIN
        A := 10;
        B := 20;
        C := A + B;
                DBMS_OUTPUT.PUT_LINE(C);
                DBMS_OUTPUT.PUT_LINE( 'sum of two numbers' || C);
END;
/
```

Output:

```
 30
sum of two numbers 30
PL/SQL procedure successfully completed.
```

Q2: Write a PL/SQL block which accepts employee number and increment is salary by 1000?

```
DECLARE
        A number(4);
        A := &Empno;
                Update emp set sal = sal + 1000 where Empno = A;
END;
```

Q3: Write a PL/SQL block which accepts empno and delete that row from the emp table?

```
DECLARE
        A number(4);
BEGIN
        A := &Empno;
        Delete from emp where Empno = A;
END;
/
```

RESULT:

Thus the PL/SQL block to satisfy some conditions by accepting input from the user has been verified and executed successfully.

Ex: No: 07 WRITE A PL/SQL BLOCK THAT HANDLES ALL TYPES OF EXCEPTIONS.

AIM:

To Write a PL/SQL block that handles all types of exceptions.

Exception Handling:

Error handling in PL / SQL is termed as exception. An exception is raised when a PL / SQL program violates Oracle rule. In such a case the normal execution stops and the control is immediately transferred to exception handling part of PL / SQL block. There are two types of exceptions. They are,

- Predefined Exceptions
- User-defined Exceptions

Predefined Exceptions are raised automatically by the system, whereas User-defined exceptions must be raised explicitly using *raise* statement.

Predefined Exception	Explanation
NO_DATA_FOUND	This exception is raised when select statement returns no records.
CURSOR_ALREADY_OPEN	This exception is raised when we try to open a cursor which is already opened.
DUP_VAL_ON_INDEX	This exception is raised when we insert duplicate values in a column, which is defined as unique_index
STORAGE_ERROR	This exception is raised if PL / SQL run out of memory or if the memory is corrupted.
PROGRAM_ERROR	This exception is raised if PL / SQL has some internal problems.
ZERO_DIVIDE	This exception is raised when we try to divide a number by zero.
INVALID_CURSOR	This exception is raised when we violate cursor operation or when we try to close a cursor which is not opened.
LOGIN_DENIED	This exception is raised when we try to enter Oracle using invalid user name or password.
INVALID_NUMBER	This exception is raised if the conversion of a

character to number fails because the string

	does not represent a valid numbers. For e.g. inserting 'ABC' in a column of datatype number will raise this exception.
TOO_MANY_ROWS	This exception is raised when the select statement returns more than one row.

Program to handle divide by zero exception:

```
declare
        a number(5);
        b number(5);
        c number(6,2);
begin
        a:=&a;
        b:=&b;
        c:=a/b;
        dbms_output.put_line(c);
exception
        when  ZERO_DIVIDE then
        dbms_output.put_line('Divisor cannot be zero');
end;
```

Output:

```
Enter value for a: 4
old   6: a:=&a;
new   6: a:=4;
Enter value for b: 0
old   7: b:=&b;
new   7: b:=0;
Divisor cannot be zero

Enter value for a: 20
old   6: a:=&a;
new   6: a:=4;
Enter value for b: 5
old   7: b:=&b;
new   7: b:=0;
4
```

Program to accept sno from supplier table and print name of supplier if the status is greater than 20 else raise exception:

```
declare
        sn supplier.sno%type;
        snam supplier.sname%type;
        stat supplier.status%type;
        e1 exception;
begin
        sn:='&serialno';
        select sname,status into snam,stat from supplier where
                                        sno=sn;
        if (stat>20) then
                dbms_output.put_line(snam);
```

```
else
```

```
                raise e1;
        end if;
exception
        when e1 then
                dbms_output.put_line('Status<=20');
        end;
```

Output:

```
Enter value for sn: S3
old   7:  sn:='&sn';
new   7:  sn:='S3';
Status<=20

Enter value for sn: S5
old   7:  sn:='&sn';
new   7:  sn:='S5';
Adams
```

Program to accept sno from supplier table and print name of supplier if he resides in LONDON else raise exception:

```
declare
        sn supplier.sno%type;
        snam supplier.sname%type;
        stat supplier.status%type;
        cit supplier.city%type;
        e1 exception;
begin
        sn:='&sn';
        select sname,status,city into snam,stat,cit from supplier
                                                        where sno=sn;
        if (cit='LONDON') then
                dbms_output.put_line(snam);
        else
                raise e1;
        end if;
exception
        when e1 then
                dbms_output.put_line('Supplier not in London');
end;
```

Output:

```
Enter value for sn: S1
old   8:  sn:='&sn';
new   8:  sn:='S1';
SMITH

Enter value for sn: S2
old   8:  sn:='&sn';
new   8:  sn:='S2';
Supplier not in London
```

RESULT:

Thus the PL/SQL block that handles all types of exceptions has been verified and executed

successfully.

Ex: No: 08 **CREATION OF PROCEDURES.**

AIM:

To work with PL / SQL Procedures

SYNTAX

```
CREATE [OR REPLACE] PROCEDURE name [(parameter[,parameter, ...])]
{IS|AS}
        [local declarations]
        BEGIN
                executable statements
        [EXCEPTION
                exception handlers]
        END [name];
```

EXECUTION:

```
SQL> SET SERVEROUTPUT ON
```

PROCEDURE FOR GCD NUMBERS

III) PROGRAM:

```
SQL> create or replace procedure pro
        is
                a  number(3);
                b  number(3);
                c  number(3);
                d number(3);
        begin
                a:=&a;
                b:=&b;
                        if(a>b) then
                                c:=mod(a,b);
                                if(c=0) then
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(b);
                                else
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(c);
                                end if;
                        else
                                d:=mod(b,a);
                                if(d=0) then
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(a);
                                else
                                        dbms_output.put_line('GCD is');
```

```
                                        dbms_output.put_line(d);
                                end if;
                        end if;
        end;
/
```

Out put:

```
Enter value for a: 8
old 8: a:=&a;
new 8: a:=8;

Enter value for b: 16
old 9: b:=&b;
new 9: b:=16;

Procedure created.

SQL> set serveroutput on;

SQL> execute pro;
        GCD is
        8
```

RESULT:

Thus the implementation of PL/SQL procedure has been verified and executed successfully.

Ex: No: 09 Creation of Database Triggers and functions

AIM:

To work with PL/SQL Triggers for the purpose of monitor the database object(table..etc) and functions .

DATABASE TRIGGERS

Triggers are similar to procedures or functions in that they are named PL/SQL blocks with declarative, executable, and exception handling sections. A trigger is executed implicitly whenever the triggering event happens. The act of executing a trigger is known as firing the trigger.

USE OF TRIGGERS

- Maintaining complex integrity constraints not possible through declarative constraints enable at table creation.
- Auditing information in a table by recording the changes made and who made them.
- Automatically signaling other programs that action needs to take place when chages are made to a table.
- Perform validation on changes being made to tables.
- Automate maintenance of the database.

TRIGGER SYNTAX

```
CREATE [OR REPLACE] TRIGGER trigername
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE [OF columns]}
[OR {DELETE | INSERT |UPDATE [OF columns]}]...
ON table
[FOR EACH ROW [WHEN condition]]
[REFERENCING [OLD AS old] [NEW AS new]]
PL/SQL block
```

Q1: Program to create a DB trigger before insert for each row on the spj table not allowing insertion for sno 's3' and pno 'p4'

```
create or replace trigger t1 before insert on spj for each row
begin
        if(:new.sno='s3' and :new.pno='p4') then
                raise_application_error(-20000,'Cannot insert s3,p4');
        end if;
end;
```

Output:

```
SQL>@p1001.sql
Trigger created
SQL>insert into spj values('s3','p4','j2',30);
insert into spj values('s3','p4','j2',30)
            *
ERROR at line 1:
ORA-20000: Cannot insert s3,p4
ORA-06512: at "309038.T1",line 4
ORA-04088: error during execution of trigger '309038.T1'
```

Q2: Program to create a DB trigger to update the qty if qty is greater than existing qty

```
create or replace trigger t2 before update on spj for each row
begin
        if(:new.qty<:old.qty) then
                raise_application_error(-20001,'Cannot Update');
        end if;
end;
```

Output:

```
SQL>@p1002.sql
Trigger created

SQL>update spj set qty=10 where sno='s1';
update spj set qty=10 where sno='s1'
   *
ERROR at line 1:
ORA-20001: Cannot update
ORA-06512: at "309038.T2",line 4
ORA-04088: error during execution of trigger '309038.T2'
```

Q3: Program to create a DB trigger not allowing deletion in supplier table

```
create or replace trigger t3 before delete on supplier for each row
begin
        raise_application_error(-20002,'Deletion Not allowed');
end;
```

Output:

Q4: Create a PL/SQL trigger which prevents the insertion of new record into the employee table.

```
create or replace trigger trig before insert on emp for each row
begin
raise_application_error (-20998, 'insertion not allowed');
end;
```

OUTPUT:

```
SQL> @karthi.sql
  5  /

Trigger created.

SQL>  insert into emp values(200,'jone','3-jun-99','man',542,10,91210);
 insert into emp values(200,'jone','3-jun-99','man',542,10,91210)



         *
ERROR at line 1:
ORA-20998: insertion not allowed
ORA-06512: at "96043.TRIG", line 2
ORA-04088: error during execution of trigger '96043.TRIG'
```

Q5: Create a PL/SQL trigger which prevents all DML operations on the table account.

```
create or replace trigger kkks before insert or delete or update on account for each row
begin
raise_application_error(-04098,'changes not allowed');
end;
```

OUTPUT:

```
SQL> @karthi1.sql
  6  /
Trigger created.
SQL> insert into account values('karthik',105);
insert into account values('karthik',105)
         *
ERROR at line 1:
ORA-20001: changes not allowed
ORA-06512: at "96043.KKK", line 2
ORA-04088: error during execution of trigger '96043.KKK'
```

Q6: Create a PL/SQL trigger to update the salary of employee if the salary is greater than the existing salary.

```
create or replace trigger kkt before update on emp for each row
begin
if :new.salary<:old.salary then
raise_application_error(-20002,'salary cannot deduce');
end if;
end;
```

OUTPUT:

```
SQL> @karthi2.sql
  7  /
Trigger created.
SQL> update emp set salary=salary-500 where ename='smith';
update emp set salary=salary-500 where ename='smith'
       *
ERROR at line 1:
ORA-20002: salary cannot deduce
ORA-06512: at "96043.KKT", line 3
ORA-04088: error during execution of trigger '96043.KKT'
```

FUNCTIONS

A function is similar to procedure, except that it returns a value. The calling program should use the value returned by the function.

CREATE FUNCTION

The **create function** command is used to create a stored function.

SYNTAX:

```
CREATE [OR REPLACE] FUNCTION name
    [(parameter[,parameter, ...])]
RETURN datatype
{IS | AS}
[local declarations]
    BEGIN
        executable statements
    RETURN value;
    [EXCEPTION
        exception handlers]
END [name];
```

Q 1: To write a PL/SQL block to find factorial of given number using function

```
declare
        n number(3);
        f number(3);
begin
        n:=&limit; f:=fact(n);
        dbms_output.put_line(n||'! = '||f);

end;

create or replace function fact(limit number) return number is
        ans number(3);
        I number(3);
begin
        ans:=1
        for i in 1..limit loop
                ans:=ans*i;
        end loop;
        return(ans);
end;
```

Output:

```
SQL>@fact.sql
Function created
SQL>@p0902.sql
Enter value for limit: 4
old   7:  n:=&limit;
new   7:  n:=4';
4! = 24
```

Q2: Create a function which count total no.of employees having salary less than 6000.

```
/*function body*/
Create or replace function count_emp(esal number)return number as
        Cursor vin_cur as Select empno,sal from emp;
        Xno emp.empno%type;
        Xsal emp.sal%type;
        C number;
Begin
        Open vin_cur;
        C:=0;
        loop
                fetch vin_cur into xno,xsal;
                        if(xsal<esal) then
                                c:=c+1;
                        end if;
                exit when vin_cur%notfound;
        end loop;
        close vin_cur;
        return c;
end;
/
Function created.
/*function specification*/
Declare
        Ne number;
        Xsal number;
Begin
        Ne:=count_emp(xsal); Dbms_output.put_line(xsal);
        Dbma_output.put_line('there are '||ne||';employees');

End;
/
```

OUTPUT

There are 8 employees.

Q3: Program to accept pno from parts table and print name of parts (using function):

```
        declare
                pn parts.pno%type:='&pno';
                pnam parts.pname%type;
        begin
                pnam:=findname(pn);
                dbms_output.put_line('Pno : '||pn||'  Pname : '||pnam);
        end;

        create or replace function findname(p parts.pno%type) return varchar is
                a parts.pname%type;
        begin
                select pname into a from parts where pno=p;
                return a;
        end;
```

Output:

```
SQL>@findname.sql
```

Function created.

```
SQL>@p0903.sql
Enter value for pno: P1
old   8:  sn:='&pno';
new   8:  sn:='P1';
Pno : P1  Pname : Nut
```

Result:

Thus the implementation of functions and database triggers has been executed successfully.

Ex: No: 11 # Database Connectivity with Front End Tools

EB BILL PREPARATION

AIM

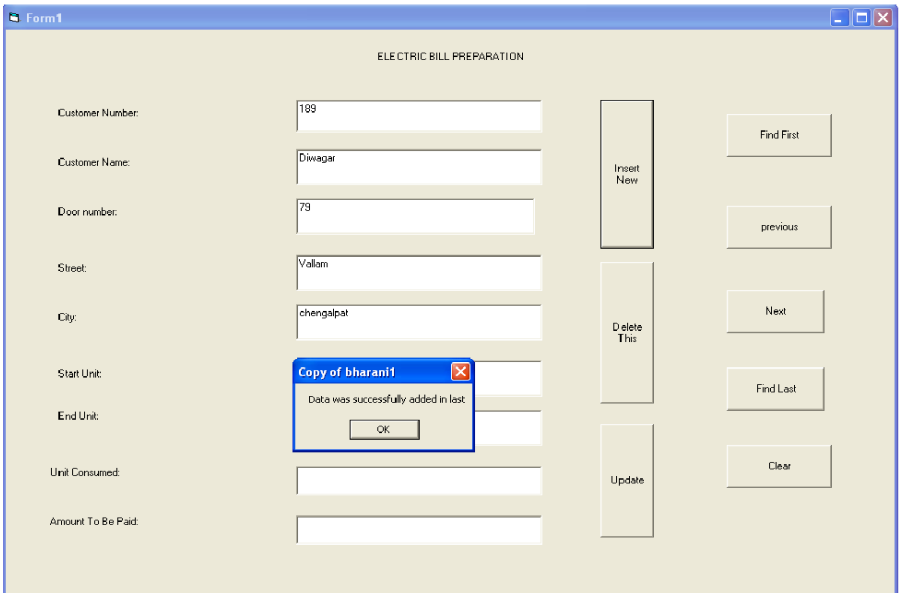
To prepare a form in VB to generate EB bill and connect it SQL back end.

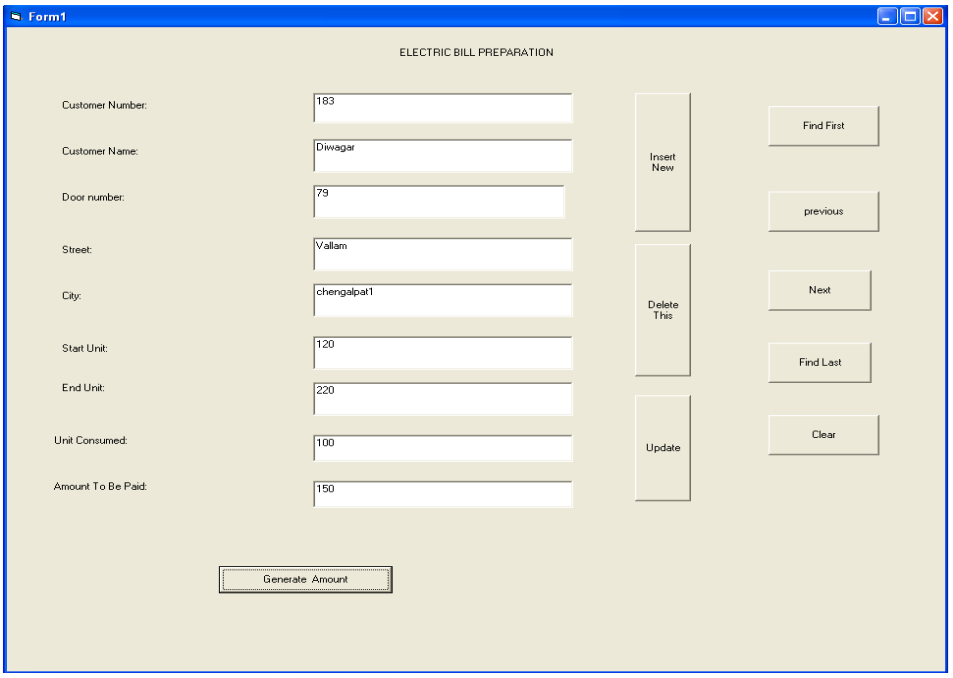
CODING

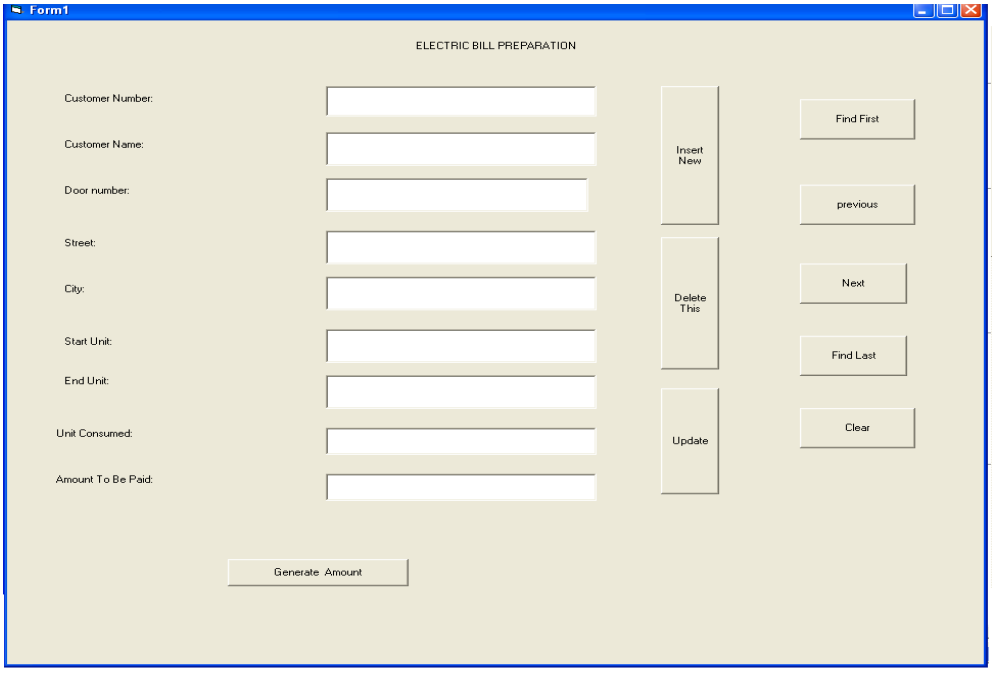
```
Dim su As Integer
Dim eu As Integer
Dim consumed As Integer
Dim amount As Integer
Dim var1 As Integer
Dim cn As New ADODB.Connection
Dim rs As New ADODB.Recordset

Private Sub clear_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
Text7.Text = ""
Text8.Text = ""
Text9.Text = ""
End Sub

Private Sub Delete_Click()
rs.Delete
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
rs.update
MsgBox " Record Deleted "
End Sub

Private Sub eb_Click()
DataReport1.Show
End Sub


Private Sub first_Click()
rs.MoveFirst
Text1.Text = rs.Fields(0)
Text2.Text = rs.Fields(1)
Text3.Text = rs.Fields(2)
```

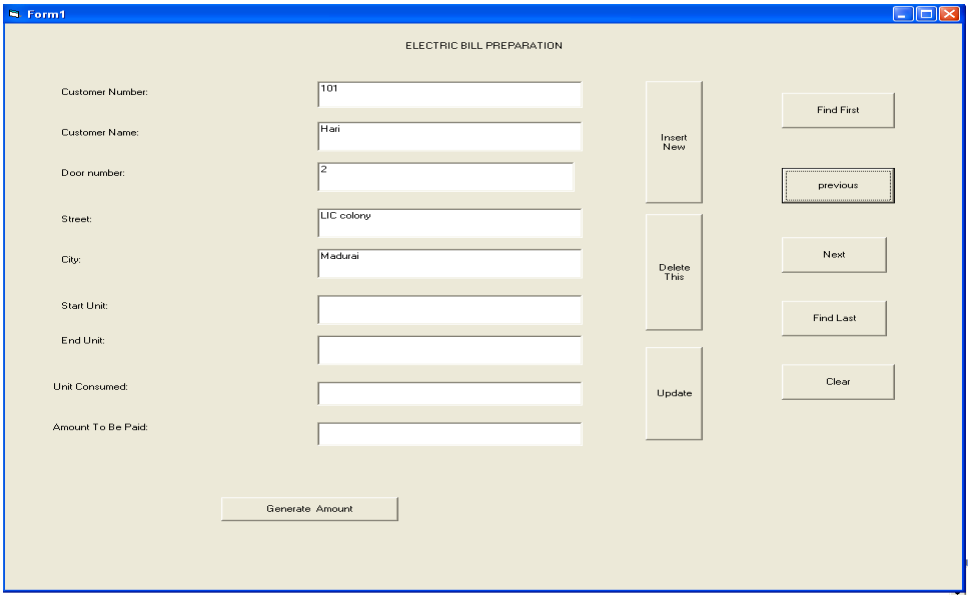
```
Text4.Text = rs.Fields(3)
Text5.Text = rs.Fields(4)
MsgBox "this is first record"
End Sub


Private Sub gm_Click()
su = Text6.Text
eu = Text7.Text
consumed = eu - su
Text8.Text = consumed
If (consumed > 400) Then
amount = consumed * 2
Else
amount = consumed * 1.5
End If
Text9.Text = amount
End Sub

Private Sub insert_Click()
rs.AddNew
rs.Fields(0) = Text1.Text
rs.Fields(1) = Text2.Text
rs.Fields(2) = Text3.Text
rs.Fields(3) = Text4.Text
rs.Fields(4) = Text5.Text
rs.update
MsgBox "Data was successfully added in last"
End Sub

Private Sub last_Click()
rs.MoveLast
Text1.Text = rs.Fields(0)
Text2.Text = rs.Fields(1)
Text3.Text = rs.Fields(2)
Text4.Text = rs.Fields(3)
Text5.Text = rs.Fields(4)
MsgBox "this is last record"
End Sub


Private Sub next_Click()
rs.MoveNext
If (rs.EOF) Then
MsgBox "this is last record"
Else
Text1.Text = rs.Fields(0)
Text2.Text = rs.Fields(1)
Text3.Text = rs.Fields(2)
Text4.Text = rs.Fields(3)
Text5.Text = rs.Fields(4)
End If
End Sub
Private Sub previous_Click()
```

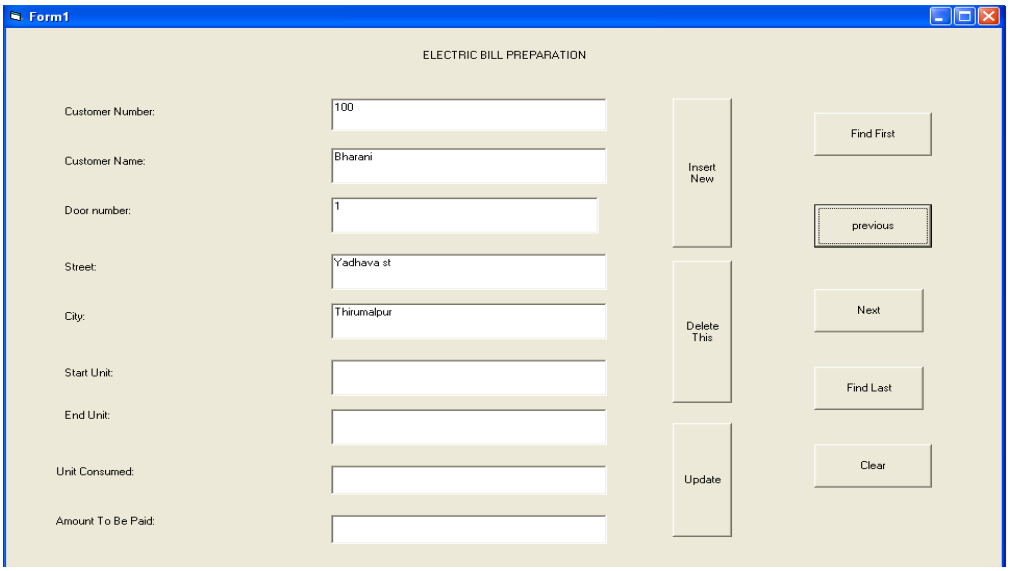
```
rs.MovePrevious
If (rs.BOF) Then
MsgBox "this is the first record"
Else
Text1.Text = rs.Fields(0)
Text2.Text = rs.Fields(1)
Text3.Text = rs.Fields(2)
Text4.Text = rs.Fields(3)
Text5.Text = rs.Fields(4)
End If
End Sub
Private Sub Form_Load()
Set cn = New ADODB.Connection
Set rs = New ADODB.Recordset
cn.Open "dsn=ebpro;User Id=96017;Password=96017;"
rs.Open "eb", cn, adOpenDynamic, adLockOptimistic
End Sub


Private Sub report_Click()
DataReport1.Show
End Sub

Private Sub update_Click()
rs.Fields(0) = Text1.Text
rs.Fields(1) = Text2.Text
rs.Fields(2) = Text3.Text
rs.Fields(3) = Text4.Text
rs.Fields(4) = Text5.Text
rs.update
MsgBox "Data Updated"

End Sub
```

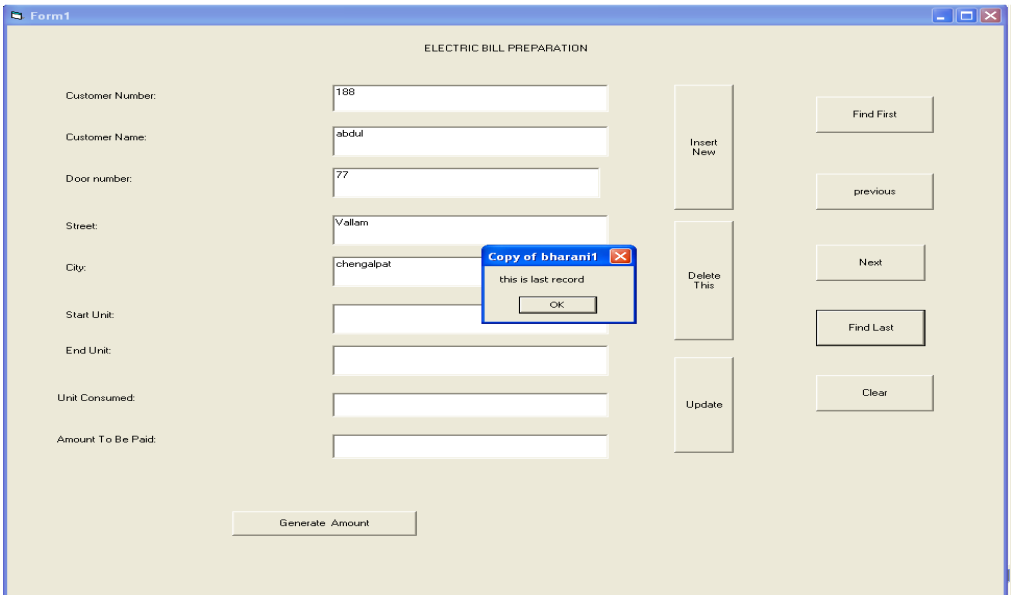


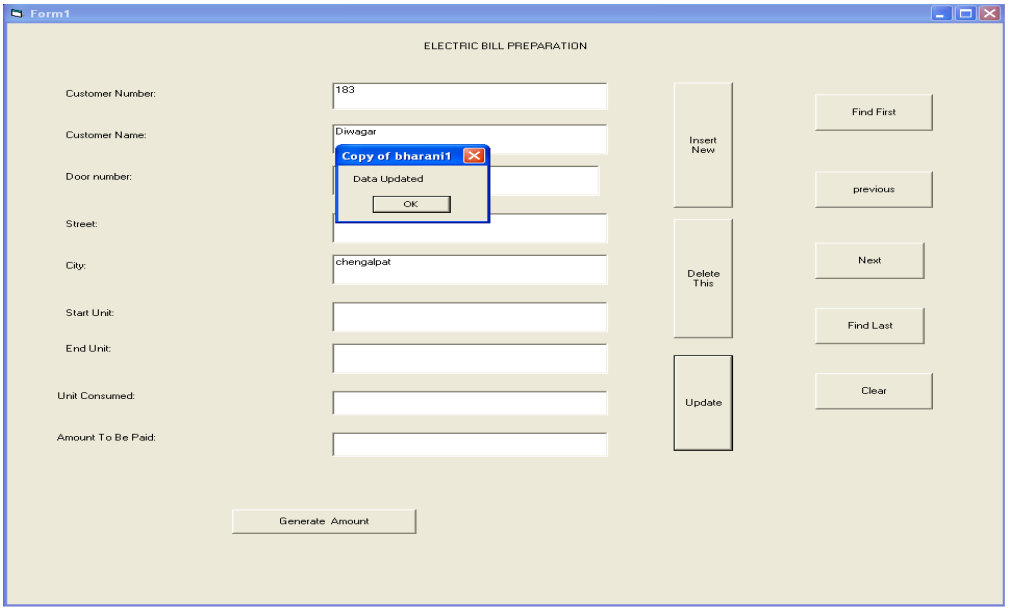

Form1
ELECTRIC BILL PREPARATION
Customer Number:
183
Customer Name:
Diwagar
Door number:
79
Street:
Vallam
City:
chengalpat1
Start Unit:
120
End Unit:
220
Unit Consumed:
100
Amount To Be Paid:
150
Generate Amount
Insert New
Delete This
Update
Find First
previous
Next
Find Last
Clear


Form1
ELECTRIC BILL PREPARATION
Customer Number:
Customer Name:
Door number:
Street:
City:
Start Unit:
End Unit:
Unit Consumed:
Amount To Be Paid:
Generate Amount
Insert New
Delete This
Update
Find First
previous
Next
Find Last
Clear


Form1
ELECTRIC BILL PREPARATION
Customer Number:
101
Customer Name:
Hari
Door number:
2
Street:
LIC colony
City:
Madurai
Start Unit:
End Unit:
Unit Consumed:
Amount To Be Paid:
Generate Amount
Insert New
Delete This
Update
Find First
previous
Next
Find Last
Clear


Form1
ELECTRIC BILL PREPARATION
Customer Number:
100
Customer Name:
Bharani
Door number:
1
Street:
Yadhava st
City:
Thirumalpur
Start Unit:
End Unit:
Unit Consumed:
Amount To Be Paid:
Insert New
Delete This
Update
Find First
previous
Next
Find Last
Clear





RESULT

Thus the mini project for eb bill with sql back end was created successfully.

Mini Project :

Bank Database :

1.find the names of all the customers, loan number,loan amount for all the customers who have a loan at the bank?

SQL>select cname,loan44.lno,amount from borrower44,loan44 where borrower44.lno=loan44.lno;

CNAME LNO AMOUNT

---------- ---------- ----------

smith 11 900

jackson 14 1500

hayes 15 1500

adams 16 1300

jones 17 1000

williams 17 1000

smith 23 2000

curry 93 500

8 rows selected.

2.find the customer name, loan number,loan amount for all the customers who have a loan at the perryridge branch?

SQL>select cname,loan44.lno,amount from borrower44,loan44 where borrower44.lno=loan44.lno and bname='perryridge';

CNAME LNO AMOUNT

---------- ---------- ----------

hayes 15 1500

adams 16 1300

3.find tha largest account balance in the bank?

a)using aggregate function

SQL> select max(bal) from account44;

MAX(BAL)

900

b)without using aggregate function

SQL> select bal from account44 minus (select t.bal from account44 t,account44 s where t.bal<s.bal);

BAL

900

4.find the names of all branches that have asserts greater than atleast one branch located in Brooklyn?

SQL> select distinct t.bname from branch t,branch s where t.asserts>s.asserts and s.bcity='brooklyn';

BNAME

---

downtown

roundhill

5.find all the customers who have a loan at the perryridge branch & order all the types?

select cname,loan44.lno,amount from borrower44,loan44 where borrower44.lno=loan44.lno and loan4

4.bname='perryridge' order by cname asc;

CNAME	LNO	AMOUNT
adams	16	1300
hayes	15	1500

6.find the name of the customer who have either a loan or an account in the bank?

SQL> (select cname from depositer44)union(select cname from borrower44);

CNAME

---

adams

curry

hayes

jackson

johnson

jones

lindsay

smith

turner

williams

10 rows selected.

7.find the names of the customers who have both loan and the account?

SQL> (select cname from depositer44)intersect(select cname from borrower44);

CNAME

---

hayes

jones

smith

8.find the names of the customers who have an account but do not have a loan at the bank?

SQL>(select cname from depositer44)minus(select cname from borrower44);

CNAME

---

johnson

lindsay

turner

9.find the average account balance at the perryridge branch?

SQL> select avg(bal) from account44 where bname='perryridge';

AVG(BAL)

400

10.find the average account balance at each branch?

SQL>select bname,avg(bal) from account44 group by bname

BNAME AVG(BAL)

---------- ----------

brighton 825

downtown 500

mianus 700

perryridge 400

redwood 700

roundhill 350

6 rows selected.

11.find the number of depositers at each branch?

SQL> select bname,count(distinct cname) from depositer44,account44 where depositer44.acno=account44.acno group by bname;

BNAME COUNT(DISTINCTCNAME)

---------- --------------------

brighton 2

downtown 2

mianus 1

redwood 1

roundhill 1

12.find only those branches that have average account more than 500?

SQL> select bname,avg(bal) from account44 group by bname having avg(bal)>500;

BNAME AVG(BAL)

---------- ----------

brighton 825

mianus 700

redwood 700

13.find the average balance for each customer who lives in perryridge and has atleast one account?

SQL> select depositer44.cname,avg(bal) from depositer44,account44,customer44 where depositer44.acno=account44.acno and depositer44.cname=customer44.cname and ccity='perryridge' group by depositer44.cn

ame having count(distinct depositer44.acno)>=1;

no rows selected

14.find the names of all the customers who have both loan and an account at the branch?

select distinct cname from borrower44 where cname in(select cname from depositer44)

SQL> /

CNAME

---

hayes

jones

smith

15.find all the customers who have a loan in yhe bank but do not have an account in the bank?

select distinct cname from borrower44 where cname not in(select cname from depositer44);

CNAME

---

adams

curry

jackson

williams

16.find names of all the branches that have asserts greater than that of each branch located in perryridge?

SQL> select bname from branch where asserts>all(select asserts from branch where bcity='perryridge');

BNAME

---

brighton

downtown

mianus

northtown

perryridge

pownal

redwood

roundhill

8 rows selected.

17.find all the customers who have both an account and a loan at the perryridge branch?

SQL> select distinct cname from borrower44,loan44 where borrower44.lno=loan44.lno and bname='perryri

dge' and (cname,bname) in (select bname,cname from depositer44,account44 where depositer44.acno=a

ccount44.acno);

CNAME

---

hayes

18.find the names of all the branches that have asserts greater than those of atleast one branch located in Brooklyn?

select bname from branch where asserts>some(select asserts from branch where bcity='brooklyn')

SQL> /

BNAME

---

downtown

roundhill

19.find the names of all the customers who have an account and a loan in the bank?

SQL> select cname from borrower44,loan44 where exists (select cname from depositer44 where depositer

44.cname=borrower44.cname);

CNAME

---

hayes

jones

smith

20.find all the customers who have an account at all the branches located in Brooklyn?

```
SQL> select distinct s.cname from depositer44 s where not exists ((select bname from branch where
bcity='brooklyn') minus (select r.bname from depositer44 t,account44 r where t.acno=r.acno and s.cname
=t.cname));

CNAME
----------
johnson

SQL> select * from customer44;

CNAME      CSTREET    CCITY
---------- ---------- ----------
adams      spring     pittsfield
brooks     senator    brooklyn
curry      north      rye
glenn      sandhill   woodside
green      walnut     stamford
hayes      main       harrison
johnson    alma       paloalto
jones      main       harrison
lindsay    park       pittsfield
smith      north      rye
turner     putnam     stamford
CNAME      CSTREET    CCITY
---------- ---------- ----------
williams   nassau     princeton
12 rows selected.

SQL> select * from branch;

BNAME      BCITY        ASSERTS
---------- ---------- ----------
brighton   brooklyn     7100000
```

```
downtown   brooklyn      9000000
mianus     horseneck      400000
northtown  rye           3700000
perryridge horseneck     1700000
pownal     bennington     300000
redwood    palo alto     2100000
roundhill  horseneck     8000000
8 rows selected.
SQL> select * from loan44;
 LNO BNAME          AMOUNT
---------- ---------- ----------
        11 roundhill        900
        14 downtown        1500
        15 perryridge      1500
        16 perryridge      1300
        17 downtown        1000
        23 redwood         2000
        93 mianus           500
7 rows selected.
SQL> select * from account44;
ACNO BNAME            BAL
---------- ---------- ----------
       101 downtown         500
       102 perryridge       400
       201 brighton         900
       215 mianus           700
       217 brighton         750
       222 redwood          700
```

```
      305 roundhill        350

7 rows selected.

SQL> select * from depositer44;

CNAME           ACNO

---------- ----------

hayes           101

johnson          101

johnson          201

jones           217

lindsay          222

smith           215

turner          305


7 rows selected.

SQL> select * from borrower44;

CNAME           LNO

---------- ----------

adams            16

curry           93

hayes            15

jackson          14

jones           17

smith            11

smith            23

williams         17


8 rows selected.
```