SENGUNTHAR ENGINEERING COLLEGE

(AUTONOMOUS)

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)
Recognized Under Section 2(f) & 12(B) of the UGC Act, 1956
NAAC Accredited with 'A' Grade

TIRUCHENGODE - 637 205 NAMAKKAL (Dt) TAMILNADU

ESTD 2001

ISO 9001 REGISTERED

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

19CSE401 DATABASE MANAGEMENT SYSTEMS LABORATORY MANUAL

FOR IV SEMESTER CSE STUDENTS

COMPILED BY

Mr. PERUMALA MAHESHRAJ AP / CSE

Mrs.S. HEMALATHA AP/CSE

FACULTY INCHARGE **ACADEMIC COORDINATOR** **HoD**

DEAN-ACADEMICS **PRINCIPAL**

INDEX

S.no	Date	Name of the Experiment	Page no	Marks awarded	Signature

LIST OF EXPERIMENTS

(Any Eigth Experiments to be conducted)

1. Data Definition Commands, Data Manipulation Commands for inserting, deleting, updating and retrieving Tables and Transaction Control statements.
2. Implement Relational model to entitle an strong and weak entities.
3. Database Querying – Simple queries, Nested queries, Sub queries and Joins.
4. Views, Sequences, Synonyms.
5. Database Programming: Implicit and Explicit Cursors.
6. Procedures and Functions.
7. Triggers.
8. Exception Handling.
9. Database Design using ER modelling, normalization and Implementation for any application.
10. Database Connectivity with Front End Tools.
11. Case Study using real life database applications.

Ex No: 01 DATA DEFINITION COMMANDS, DATA MANIPULATION COMMANDS FOR INSERTING,
Date: DELETING, UPDATING AND RETRIEVING TABLES AND
TRANSACTION CONTROL STATEMENTS

Aim:

To study and execute the various categories of DDL, DML command and Transaction control statements.

DATA DEFINITION LANGUAGE:

It is used to create an table, alter the structure of an table and also drop the table. The DDL commands are,

1. Create 2. Alter 3. Describe 4. Truncate 5. Drop

CREATE COMMAND:

This command is used to create the table.

Syntax:

Create table tablename (columnname1 datatype (size), columnname2 datatype (size) … column name n datatype (size));

DESC COMMAND:

This command is used to give the structure of the table.

Syntax:

SQL>desc tablename;

ALTER COMMAND:

This is used for adding the values and also modifying the table.

Syntax:

ADD:

SQL>alter table tablename add (columnname1 datatype (size), columnname2 datatype (size) … columnnamen datatype (size));

MODIFY:

SQL>alter table tablename modify (columnname (newdatatype (newsize),…….);

TRUNCATE COMMAND:

The contents of the table are deleted by using this command.

Syntax:

SQL> truncate table tablename

DROP COMMAND:

This command is used to delete the entire table.

Syntax:

SQL> drop table tablename

CREATING NEW TABLE FROM EXISTING TABLE-AS SELECT:

Syntax:

SQL> Create table newtablename (columnname1, columnname2, columnnamen) as select columnname1, Columnnamen from oldtablename;

RENAME:

Syntax:

SQL>rename oldtablename to newtablename;

DATA MANIPULATION LANGUAGE:

DML or data modification SQL commands are the SQL commands which let the user move data in and out of a data base.

The DML commands are

Insert

Select

Update

Delete

INSERT COMMAND

It is used to insert the values into table.

Syntax 1 :

SQL>insert into tablename values (columnname1……columnnamen);

Syntax 2 :

SQL>insert into tablename values ('&column1',&column2…);

SELECT COMMAND:

This command is used to view the record in the table.

Syntax:

SQL>select * from tablename;

select columnname from tablename;select * from tablename where condition;

UPDATE COMMAND

Used to update the table values when new values are added .

Syntax:

SQL>update(table name) set (coloumnname) where (condition);

DELETE COMMAND:

Deleting the new which contain the values.

Syntax:

delete from tablename;

delete from tablename where columnname=value;

TRANSACTION CONTROL LANGUAGE:

The TCL language is used for controlling the access to the table and hence securing the database. TCL is used to provide certain privileges to a particular user. Privileges are rights to be allocated. The privilege commandsare namely,

Grant

Revoke

Commit

Savepoint

Rollback

GRANT COMMAND: It is used to create users and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to theirdatabase objects to other users.

REVOKE COMMAND: Using this command, the DBA can revoke the granted database privileges from the user.

COMMIT : It is used to permanently save any transaction into database.

SAVEPOINT: It is used to temporarily save a transaction so that you can rollback to that point whenever necessary

ROLLBACK: It restores the database to last commited state. It is also use with savepoint command to jumpto a savepoint in a transaction.

SYNTAX:

GRANT COMMAND:

Grant < database_priv [database_priv…..] > to <user_name> identified by <password> [,<password…..];

Grant <object_priv> | All on <object> to <user | public> [With Grant Option];

REVOKE COMMAND:

Revoke <database_priv> from <user [, user] >;

Revoke <object_priv> on <object> from < user | public >;

<database_priv> -- Specifies the system level priveleges to be granted to the users or roles. This includes create / alter / delete any object of the system.

<object_priv> -- Specifies the actions such as alter / delete / insert / references / execute / select / update fortables.

<all> -- Indicates all the priveleges.

[With Grant Option] – Allows the recipient user to give further grants on the objects.

The priveleges can be granted to different users by specifying their names or to all users by using the "Public" option.

COMMIT:

Commit;

SAVEPOINT:

Savepoint savapoint_name;

ROLLBACK:

Rollback to savepoint_name;

SAMPLE OUTPUT:

DDL:

Create a table called EMP with the following structure.

Name	Type
EMPNO	NUMBER(6)
ENAME	VARCHAR2(20)
JOB	VARCHAR2(10)
MGR	NUMBER(4)
DEPTNO	NUMBER(3)
SAL	NUMBER(7,2)

Allow NULL for all columns except ename and job.

Solution:

Understand create table syntax.

Use the create table syntax to create the said tables.

Create primary key constraint for each table as understand from logical table structure.

CREATE COMMAND:

SQL> create table student (name varchar2 (20), rollno number(5));Table created.

Rules:

Oracle reserved words cannot be used.

Underscore, numerals, letters are allowed but not blank space.

Maximum length for the table name is 30 characters.

2 different tables should not have same name.

We should specify a unique column name.

We should specify proper data type along with width.

We can include “not null” condition when needed. By default it is ‘null’.

DESC COMMAND:

SQL> desc student;

Name	Type
NAME	VARCHAR2 (20)
ROLLNO	NUMBER (5)

ALTER (ADD) COMMAND:

SQL> alter table student add (fees number (6));

Table altered.

SQL> desc student;

Name	Type
NAME	VARCHAR2 (20)
ROLLNO	NUMBER (5)
FEES	NUMBER (6)

ALTER (MODIFY) COMMAND:

SQL> alter table student modify (rollno number (10));

Table altered.

SQL> desc student;

Name	Type
NAME	VARCHAR2 (20)
ROLLNO	NUMBER (10)
FEES	NUMBER (6)

TRUNCATE COMMAND:

SQL> truncate table student;

Table truncated.

DROP COMMAND:

SQL> drop table student;

Table dropped.

CREATING NEW TABLE FROM EXISTING TABLE-AS SELECT:

SQL> create table stu (name, rollno, fees) as select * from student;

Table created.

SQL> desc stu;

Name	Type
NAME	VARCHAR2 (20)
ROLLNO	NUMBER (10)
FEES	NUMBER (6)

SQL> create table stu1 (name, rollno) as select name, total from student;Table created.

SQL> desc stu1;

Name	Type
NAME	VARCHAR2 (20)
ROLLNO	NUMBER (10)

RENAME:

SQL> rename stu1 to stu2;

Table renamed.

SQL> desc stu1;

ERROR: ORA-04043: object stu1 does not exist.

SQL> desc stu2;

Name	Null?	Type
NAME		VARCHAR2 (20)
ROLLNO		NUMBER (10)

DML:

CREATE COMMAND:

SQL> create table dml (sname varchar2 (15), rno number (5), course varchar (9), fees number (12));

Table created.

SQL> desc dml;

Name	Null?	Type
SNAME		VARCHAR2(15)
RNO		NUMBER(5)
COURSE		VARCHAR2(9)
FEES		NUMBER(12)

INSERT COMMAND:

SQL> insert into dml values ('&sname', &rno, '&course', &fees);Enter value for sname: anju

Enter value for rno: 101 Enter value for course: cseEnter value for fees: 40000

old 1: insert into dml values('&sname', &rno, '&course', &fees)

new 1: insert into dml values('anju',101,'cse',40000)1 row created.

SQL> **/**

Enter value for sname: keerthiEnter value for rno: 106

Enter value for course: it Enter value for fees: 50000

old 1: insert into dml values('&sname',&rno,'&course',&fees) new 1: insert into dml values ('keerthi',106,'it',50000)

1 row created.

SELECT COMMAND:

Selecting all attributes.

SQL> select * from dml;

SNAME RNO COURSE FEES

---

anju 101 cse 40000

keerthi 106 it 50000

Selecting particular attributes

SQL>select sname, rno from dm1;

SNAME RNO

---

anju 101

keerthi 106

Selecting particular record or row or tuple

SQL> select * from dml where rno=106;

SNAME RNO COURSE FEES

---

keerthi 106 it 50000

ALTER COMMAND:

SQL> alter table dml add(address varchar(17));

Table altered.

SQL> select * from dml;

SNAME RNO COURSE FEES ADDRESS

---

anju 101 cse 40000

keerthi 106 it 50000

UPDATE COMMAND:

SQL> update dml set fees=60000 where course='cse';2 rows updated

SQL> select *from dml;

SNAME RNO COURSE FEES ADDRESS

---

anju 101 cse 60000

keerthi 106 it 50000

DELETE COMMAND:

SQL> delete from dml where sname='keerth';1 row deleted.

SQL> select *from dml;

SNAME RNO COURSE FEES ADDRESS

---

anju 101 cse 60000

SQL> delete from dml;s

1 row deleted.

SQL> select *from dml;

No rows selected.

TCL:

Consider the following tables namely “DEPARTMENTS” and “EMPLOYEES”

Their schemas are as follows ,

Departments (dept _no , dept_ name, dept_location);Employees (emp_id , emp_name , emp_salary);

SQL> Grant all on employees to abcde;Grant succeeded.

SQL> Grant select , update , insert on departments to abcde with grant option;Grant succeeded.

SQL> Revoke all on employees from abcde;Revoke succeeded.

SQL> Revoke select , update , insert on departments from abcde;Revoke succeeded.

COMMIT, ROLLBACK and SAVEPOINT:

SQL> select * from class;

NAME ID

---

anu 1

brindha 2

chinthiya 3

divya 4

ezhil 5

SQL> insert into class values('gayathri',9);1 row created.

```
SQL> commit;
Commit complete.

SQL> update class set name='hema' where id='9';1 row updated.
SQL> savepoint A;
Savepoint created.

SQL> insert into class values('indu',11);1 row created.
SQL> savepoint B;
Savepoint created.

SQL> insert into class values('janani',13);1 row created.
SQL> select * from class;
      NAME            ID
   -------        -------
      Anu             1
      brindha         2
      chinthiya       3
      divya           4
      ezhil           5
      fairoz          7
      hema            9
      indu            11
      janani          13
9 rows selected.
SQL> rollback to B;
Rollback complete.
SQL> select * from class;
     NAME        ID
     ------------------------
     anu         1
     brindha     2
     chinthiya   3
     divya       4
     ezhil       5
```

fairoz	7
hema	9
indu	11

8 rows selected.

SQL> rollback to A;

Rollback complete.

SQL> select * from class;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9

RESULT:

Thus the various DML, DDL and TCL commands were studied and executed successfully.

Ex No: 02 Implement Relational model to entitle an strong and weak entities

Date:

Aim:

To Implement Relational model to entitle an strong and weak entities.

Description:

- **Implement Relational Model**: This indicates that the task involves utilizing the relational model for database management. The relational model is a method for structuring data using relations (tables) composed of rows and columns, with each row representing a record and each column representing an attribute of the record.
- **Entitle**: This term may refer to assigning titles or names to entities within the database. It could also imply establishing relationships or defining permissions for these entities.
- **Strong and Weak Entities**: In the context of database design, entities refer to objects or concepts that are represented in the database. Strong entities have a primary key attribute that uniquely identifies each instance of the entity, while weak entities rely on a foreign key in addition to the primary key of another entity for identification.

Create Tables:

```
CREATE TABLE Strong_Entity (
strong_id INT PRIMARY KEY,
strong_attribute VARCHAR(50)
);
CREATE TABLE Weak_Entity (
weak_id INT,
strong_id INT,
weak_attribute VARCHAR(50),
PRIMARY KEY (weak_id, strong_id),
FOREIGN KEY (strong_id) REFERENCES Strong_Entity(strong_id)
);
DESCRIBE Strong_Entity;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| strong_id     | INT         | NO   | PRI | NULL    |       |
```

| strong_attribute | VARCHAR(50) | YES | | NULL | |

+---------------+-------------+------+-----+---------+-------+

DESCRIBE Weak_Entity;

+-----------------+---------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-----------------+---------+------+-----+---------+-------+

| weak_id | INT | YES | PRI | NULL | |

| strong_id | INT | YES | PRI | NULL | |

| weak_attribute | VARCHAR(50) | YES | | NULL | |

+-----------------+---------+------+-----+---------+-------+

Insert Data:

```
INSERT INTO Strong_Entity (strong_id, strong_attribute)
VALUES (1, 'Strong Data');
INSERT INTO Weak_Entity (weak_id, strong_id, weak_attribute)
VALUES (100, 1, 'Weak Data');
```

Select Data:

```
SELECT * FROM Strong_Entity;
SELECT w.weak_id, s.strong_attribute, w.weak_attribute
FROM Weak_Entity w
JOIN Strong_Entity s ON w.strong_id = s.strong_id;
```

Update Data:

```
-- Update strong entity attribute
UPDATE Strong_Entity
SET strong_attribute = 'Updated Data'
WHERE strong_id = 1;
-- Update weak entity attribute
UPDATE Weak_Entity
SET weak_attribute = 'Updated Weak Data'
WHERE weak_id = 100;
```

Delete Data:

```
-- Delete from weak entity
DELETE FROM Weak_Entity
WHERE weak_id = 100;
-- Delete from strong entity
DELETE FROM Strong_Entity
```

WHERE strong_id = 1;

Sample Output:

Strong_Entity:

```
+-----------+----------------+
| strong_id | strong_attribute |
+-----------+----------------+
|    1      |   Strong Data   |
+-----------+----------------+
```

Weak_Entity:

```
+---------+-----------+----------------+
| weak_id | strong_id | weak_attribute |
+---------+-----------+----------------+
|   100   |     1     |   Weak Data    |
```

After Updating:

Strong_Entity:

```
+-----------+-----------------+
| strong_id | strong_attribute |
+-----------+-----------------+
|    1      |  Updated Data    |
+-----------+-----------------+
```

Weak_Entity:

```
+---------+-----------+--------------------+
| weak_id | strong_id |   weak_attribute    |
+---------+-----------+--------------------+
|   100   |     1     |  Updated Weak Data |
```

After Deleting:

Strong_Entity:

(empty)

Weak_Entity:

(empty)

OUTPUT:

Strong_Entity:

```
+-----------+----------------+
| strong_id | strong_attribute |
+-----------+----------------+
```

1	Apple
2	Banana
3	Orange

Weak_Entity:

weak_id	strong_id	weak_attribute
100	1	Red
101	2	Yellow
102	3	Orange

Select Data:

Strong_Entity:

strong_id	strong_attribute
1	Apple
2	Banana
3	Orange

Weak_Entity:

weak_id	strong_id	weak_attribute
100	1	Red
101	2	Yellow
102	3	Orange

After Updating Data:

Let's update the weak attribute for weak_id = 100 to "Green".

Strong_Entity:

strong_id	strong_attribute
1	Apple

2	Banana
3	Orange

Weak_Entity:

weak_id	strong_id	weak_attribute
100	1	Green
101	2	Yellow
102	3	Orange

After Deleting:

Let's delete the strong entity with strong_id = 1.

Strong_Entity:

strong_id	strong_attribute
2	Banana
3	Orange

Weak_Entity:

weak_id	strong_id	weak_attribute
101	2	Yellow
102	3	Orange

RESULT:

Thus to implement Relational model to entitle an strong and weak entities was executed successfully.

Ex no:3 DATABASE QUERYING – SIMPLE QUERIES, NESTED QUERIES, SUB QUERIES AND
Date: JOINS

Aim:

To study and execute the database queries such as simple, nested, sub queries and join operations

Description:

Join:

Join is a query in which data is retrieved from two or more table. A join matches data from two or more tables, based on value of one or more columns in each table.

Inner join

Equi join

Natural join

Cross join

Outer join

Left outer join

Right out join

Full out join

Self join

Table structure:

SQL>desc suppliers;

Name NULL? Type

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

Supplier_id number(5)

Supplier_name varchar2(25)SQL>desc order;

NAME NULL? TYPE

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

Order_id number(6)

Supplier_id number(5)

Order_date date

SQL>select * from suppliers;

Supplier:

Supid	supname
11	abi
12	chindu
13	nithi
14	selvi

SQL>select * from order;

Order:

Oid	supid	odate
111	12	1-9-09
222	23	8-9-09
333	14	6-9-09
444	25	3-9-09
555	15	17-9-09

INNER JOINS:

Inner join returns the matching rows from the tables that are being joined.

EQUI-JOIN:

An equi-join, also known as an equijoin, is a specific type of comparator-based join, or theta join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join.

SQL>select * from tablename1, tablename2 where tablename1.columnname = tablename2.columnname;

NATURAL JOIN:

A natural join offers a further specialization of equi-joins. The join predicate arises implicitly by comparing all columns in both tables that have the same column-name in the joined tables. The resulting joined table contains only one column for each pair of equally-named columns.

The above sample query for inner joins can be expressed as a natural join in the following way:
SQL>select *from employee natural join department;

CROSS JOIN:

A cross join, Cartesian join or product provides the foundation upon which all types of inner joins operate. A cross join returns the Cartesian product of the sets of records from the two joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or where the join-condition is

absent from the statement.

If A and B are two sets, then the cross join is written as A × B. The SQL code for a cross join lists the tables for joining (FROM), but does not include any filtering join-predicate.

Examples of an explicit cross join:

SQL>select *from employee cross join department;

Example of an implicit cross join:

SQL>select *from employee, department;

Syntax:

SQL>selecttablename1.columnname1…..tablename1.columnnamen, table2.columnname1…….table2.columnname2 from tablename1 innerjoin tablename2 on tablename1.columnname=tablename2.columnname;

Sample Output:

SQL>select * from supplier, order where supplier.supid=ord.supid;

Supid	supname	Oid	supid	odate
12	abi	111	12	1/9/09
14	nithi	333	14	6/9/09
15	selvi	555	15	17/9/09

SQL>select supplier.supid, supplier.supname, order.oid, order.supid from supplier innerjoin order on supplier.supid=ord.supid**;**

Supid	supname	Oid	supid
12	abi	111	12
14	nithi	333	14
15	selvi	555	15

OUTER JOIN:

An outer join is an extended from of the inner join. In this, the rows in one table having no matching rowsin the other table will also appear in the result table with null.

LEFT OUTER JOIN:

The left outer join returns matching rows from the tables being joined and also no matching rows from the left tables in the result and places null values in the attributes that come from the right table.

Syntax:

SQL> select tablename1.columnname1…..tablename1.columnnamen, table2.columnname1 …….table2.columnnamen from tablename1 left outer join tablename2 on tablename1.columnname=tablename2.columnname;

Output:

SQL>select * from stud;

NAME	MARK
Aaa	70
bbb	80
ccc	90

SQL>select * from std;

NAME	RANK
aaa	3
bbb	2
ddd	1

SQL>select stud.name, stud.marks, std.rank from stud left outer join std on stud.name=std.name;

NAME	MARKS	RANK
aaa	70	3
bbb	80	2
ccc	90	1

RIGHT OUTER JOIN:

The right outer join operation returns matching rows from the tables being joined, and also non matching rows from the right table in the result and places null values in the attributes that comes from the left table.

Syntax:

SQL> select tablename1.columnname1…..tablename1.columnnamen, table2.columnname1 ……. table2.columnname2 from tablename1 right outer join tablename2 on tablename1.columnname = tablename2.columnname;

Output:

SQL>select stud.name, stud.marks, std.rank from stud right outer join std on stud.name=std.name;

NAME	MARKS	RANK
aaa	70	3
bbb	80	2

FULL OUTER JOIN:

It includes all tuples from left relation that do not match with any tuples in right relation as well as tuples from right relation that do not match with any tuples in left relation and adds them to resultant location.

Syntax:

SQL> select tablename1.columnname1…..tablename1.columnnamen, table2.columnname1 ……. table2.columnname2 from tablename1 full outer join tablename2 on tablename1.columnname = tablename2.columnname;

Output:

SQL>select stud.name, stud.marks, std.rank from stud full outer join std on stud.name=std.name;

NAME	MARKS	RANK
aaa	70	3
bbb	80	2
ccc	90	1

SELF-JOIN

A self-join is joining a table to itself.

SYNTAX:

SQL>select column(s) from <table1><table2>where<condition>

EXAMPLE:

SQL>select dhar.fname “employee”, him.lastname “manager” from emp_master dharm, emp_master seraphwhere dharm.mgr_no=sereph.emp_no;

Nested Queries:

Nesting of queries one within another is known as a nested queries.

Subqueries

The query within another is known as a subquery. A statement containing subquery is called parent statement.The rows returned by subquery are used by the parent statement.

Example: select ename, eno, address where salary >(select salary from employee where ename ='jones');

Types

Subqueries that return several values

Subqueries can also return more than one value.such results should be made use along with the operators in andany.

Example: select ename, eno, from employee where salary <any (select salary from employee where deptno =10');

Multiple queries

Here more than one subquery is used. These multiple subqueries are combined by me

Ans: of 'and' & 'or' keywords.

Correlated subquery

A subquery is evaluated once for the entire parent statement whereas a correlated subquery is evaluated once perrow processed by the parent statement.

Example: select * from emp x where x.salary > (select avg(salary) from emp where deptno =x.deptno);

Above query selects the employees details from emp table such that the salary of employee is > the average salaryof his own department.

Sample output:

Select all employees from 'maintenance' and 'development' dept.

Solution:

1. Use select from where clause with the from coming from emp and dept tables and a condition joining these twotable using the key deptno which connects both the tables with an equality condition.

Ans:

SQL> select ename from empd where deptno in(select deptno from deptd where dname in('MAINTAINANCE','DEVELOPMENT'));

ENAME

SMITH

ASANT

ALLEN

WARD

JONES

BLAKE

FORD

ALLEY

DRANK

9 rows selected.

Display all employee names and salary whose salary is greater than minimum salary of the company and job title starts with ‘M’.

Solution:

Use select from clause.

Use like operator to match job and in select clause to get the result.

Ans:

```
SQL> select ename , sal from empd where job like 'M%' and sal>(select min(sal) from empd);
ENAME          SAL
 ---------- -------

JONES          5975
BLAKE          9850
```

Issue a query to find all the employees who work in the same job as jones.Ans:

```
SQL> select ename, job from empd where job=(select job from empd where ename='JONES')
ENAME          JOB
 ---------- ---------

JONES          MANAGER
BLAKE          MANAGER
```

Issue a query to display information about employees who earn more than any employee in dept 30.Ans:

```
SQL> select * from empd where sal>(select max(sal) from empd where deptno=30);
EMPNO ENAME   JOB         MGR    DEPTNO  SAL COMM          DOB
 ------ --------- ----------------------------------------

7839    CLARK  CEO  10        9900    1000 16-MAR-72
```

Display the employees who have the same job as jones and whose salary >= fords.Ans:

```
SQL> select ename from empd where job=(select job from empd where ename='JONES') and sal>(select
sal fromempd where ename='FORD');
ENAME

 ----------

JONES
BLAKE
```

Write a query to display the name and job of all employees in dept 20 who have a job that someone in the Management dept as well.

Ans:

SQL> select ename, job from empd where deptno in (20, (select deptno from deptd where dname='MANAGEMENT);

ENAME JOB

SMITH CLERK

ASANT SALESMAN

JONES MANAGER

FORD SUPERVISOR

Issue a query to list all the employees who salary is > the average salary of their own dept. Ans:

SQL> select ename from empd where sal>(select avg(sal) from empd where deptno in (10,20,30));

ENAME

JONES

BLAKE

CLARK

Write a query that would display the empname, job where each employee works and the name of their dept.

SQL> select empd.ename,empd.job,deptd.dname from empd,deptd where empd.deptno=deptd.deptno;

ENAME	JOB	DNAME
SMITH	CLERK	DEVELOPMENT
ASANT	SALESMAN	DEVELOPMENT
ALLEN	SALESMAN	MAINTAINANCE
WARD	SALESMAN	MAINTAINANCE
JONES	MANAGER	DEVELOPMENT
BLAKE	MANAGER	MAINTAINANCE
SCOTT	HOD	MANAGEMENT
CLARK	CEO	MANAGEMENT

FORD	SUPERVISOR	DEVELOPMENT
ALLEY	SALESMAN	MAINTAINANCE
DRANK	CLERK	MAINTAINANCE

Write a query to list the employees having the same job as employees located in ‘ mainblock’.(use multiple subquery)

SQL> select empd.ename, empd.job from empd,deptd where empd.deptno=deptd.deptno and job in(selectempd.job from empd,dept where empd.deptno=deptd.deptno and deptd.loc='MAINBLOCK');

ENAME	JOB
ALLEN	SALESMAN
WARD	SALESMAN
BLAKE	MANAGER
SCOTT	HOD
CLARK	CEO
ALLEY	SALESMAN
DRANK	CLERK

7 rows selected.

Write a query to list the employees in dept 10 with the same job as anyone in the development dept.

SQL> select empd.ename from empd,deptd where empd.deptno=deptd.deptno and empd.deptno=10 and job in (select empd.job from empd,deptd where empd.deptno=deptd.deptno and deptd.dname='DEVELOPMENT')

No rows selected

Write a query to list the employees with the same job and salary as ‘ford’.Ans:

select ename from empd where sal=(select sal from empd where ename='FORD') and job=(select job from empdwhere ename='FORD') and ename not like 'FORD';

No rows selected

Write a query to list the employees in dept 20 with the same job as anyone in dept 30.Ans:

SQL> select ename from empd where deptno=20 and job in (select job from empd where deptno=30);

ENAME
SMITH
JONES
ASANT

List out the employee names who get the salary greater than the maximum salaries of dept with dept no 20,30

Solution:

Use select from clause.

Use any operator in select clause to get the result.

SQL> select ename from empd where sal>(select max(sal) from empd where deptno in(20,30))

ENAME

CLARK

Display the maximum salaries of the departments whose maximum salary is greater than 9000.Solution:

Use select from clause.

Use group by and having functions on name in select clause to get the result.

SQL> select deptno from empd where sal>9000 group by deptno

DEPTNO

10

30

Display the maximum salaries of the departments whose minimum salary is greater than 1000 and lesser than 5000.

Solution:

Use select from clause.

Use group by and having functions on name in select clause to get the result.

SQL> select max(sal),deptno from empd where deptno in (select deptno from empd where sal between 100);

MAX(SAL) DEPTNO

9900 10

5975 20

9850 30

JOINS

Create the following table :AccDept.(Accredited Department by quality council)

DEPTNO	DNAME	DCity
10	MANAGEMENT	MAIN BLOCK
20	DEVELOPMENT	MANUFACTURING UNIT
30	MAINTAINANCE	MAIN BLOCK

EQUI-JOIN

Display the departments that are accredited by the quality council. Solution:

Use select from clause.

Use equi join in select clause to get the result.

Ans:

```
SQL> select deptno,dname from ad
DEPTNO      DNAME
-------     ----------
10      MANAGEMENT
20      DEVELPOMENT
30      MAINTAINANCE
```

NON-EQUIJOIN

Display the employees of departments which are not accredited by the quality council Solution:

Use select from clause.

Use non equi join in select clause to get the result.

Ans:

```
SQL> select ename from empd where deptno not in ( select deptno from ad);no rows selected
```

RESULT:

Thus the database querying – simple queries, nested queries, sub queries and joins were studied and executed successfully.

Ex. No : 04 **VIEWS, SEQUENCES, SYNONYMS**

Date :

AIM:

To create views, synonyms and sequences using DDL, DML and DCL statements.

DESCRIPTION:

Views:

A database view is a logical or virtual table based on a query. It is useful to think of a view as a stored query. Views are queried just like tables.

A DBA or view owner can drop a view with the DROP VIEW command.

TYPES OF VIEWS:

Updatable views – Allow data manipulation

Read only views – Do not allow data manipulation

TO CREATE THE TABLE 'FVIEWS':

SQL> create table fviews(name varchar2(20),no number(5), sal number(5), dno number(5)); Table created.

SQL> insert into fviews values('xxx',1,19000,11);1 row created.

SQL> insert into fviews values('aaa',2,19000,12);1 row created.

SQL> insert into fviews values('yyy',3,40000,13);1 row created.

SQL> select * from fviews;

NAME	NO	SAL	DNO
xxx	1	19000	11
aaa	2	19000	12
yyy	3	40000	13

TO CREATE THE TABLE 'DVIEWS':

SQL> create table dviews(dno number(5), dname varchar2(20));

Table created.

SQL> insert into dviews values(11,'x');

1 row created.

SQL> insert into dviews values(12,'y');

1 row created.

SQL> select * from dviews;

DNO	DNAME
11	x
12	y

CREATING THE VIEW ‘SVIEW’ ON ‘FVIEWS’ TABLE:

SQL> create view sview as select name,no,sal,dno from fviews where dno=11;View created.

SQL> select * from sview;

NAME	NO	SAL	DNO
xxx	1	19000	11

Updates made on the view are reflected only on the table when the structure of the table and the view are notsimilar -- proof

SQL> insert into sview values ('zzz',4,20000,14);1 row created.

SQL> select * from sview;

NAME	NO	SAL	DNO
xxx	1	19000	11

SQL> select * from fviews;

NAME	NO	SAL	DNO
xxx	1	19000	11
aaa	2	19000	12
yyy	3	40000	13
zzz	4	20000	14

Updates made on the view are reflected on both the view and the table when the structure of the table and theview are similar – proof

CREATING A VIEW ‘IVIEW’ FOR THE TABLE ‘FVIEWS’:

SQL> create view iview as select * from fviews;View created.

SQL> select * from iview;

NAME NO SAL DNO

---

xxx 1 19000 11

aaa 2 19000 12

yyy 3 40000 13

zzz 4 20000 14

PERFORMING UPDATE OPERATION:

SQL> insert into iview values ('bbb',5,30000,15);1 row created.

SQL> select * from iview;

NAME NO SAL DNO

---

xxx 1 19000 11

bbb 5 30000 15

SQL> select * from fviews;

NAME NO SALDNO

---

xxx 1 19000 11

aaa 2 19000 12

yyy 3 40000 13

zzz 4 20000 14

bbb 5 30000 15

CREATE A NEW VIEW 'SSVIEW' AND DROP THE VIEW:

SQL> create view ssview(cusname,id) as select name, no from fviews where dno=12;View created.

SQL> select * from ssview;

CUSNAME ID

---

aaa 2

SQL> drop view ssview;

View dropped.

TO CREATE A VIEW 'COMBO' USING BOTH THE TABLES 'FVIEWS' AND 'DVIEWS':

SQL> create view combo as select name,no,sal,dviews.dno,dname from fviews,dviews where fviews.dno=dviews.dno;

View created.

SQL> select * from combo;

NAME NO SAL DNO DNAME

---

xxx 1 19000 11 x

aaa 2 19000 12 y

TO PERFORM MANIPULATIONS ON THIS VIEW:

SQL> insert into combo values('ccc',12,1000,13,'x');insert into combo values('ccc',12,1000,13,'x')

Examples:

SQL> select * from class;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9

rows selected.

Create synonym:

SQL> create synonym c1 for class;

Synonym created.

SQL> insert into c1 values('kalai',20);1 row created.

SQL> select * from class;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4

ezhil	5
fairoz	7
hema	9
kalai	20

rows selected.

SQL> select * from c1;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9
kalai	20

rows selected.

SQL> insert into class values('Manu',21);1 row created.

SQL> select * from c1;

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7
hema	9
kalai	20
Manu	21

rows selected.

Drop Synonym:

SQL> drop synonym c1;

Synonym dropped.

SQL> select * from c1;select * from c1

* ERROR at line 1:

ORA-00942: table or view does not exist

Sequences:

Oracle provides the capability to generate sequences of unique numbers, and they are called **sequences.** Just like tables, views, indexes, and synonyms, a sequence is a type of database object.

Sequences are used to generate unique, sequential integer values that are used as primary key values indatabase tables.

The sequence of numbers can be generated in either ascending or descending order.

Creation of table:

SQL> create table class(name varchar(10),id number(10));Table created.

Insert values into table:

SQL> insert into class values('&name',&id);Enter value for name: anu

Enter value for id: 1

old 1: insert into class values('&name',&id)new 1: insert into class values('anu',1)

1 row created.

SQL> /

Enter value for name: brindha

Enter value for id: 02

old 1: insert into class values('&name',&id)new 1: insert into class values('brindha',02)

1 row created.

SQL> /

Enter value for name: chinthiya

Enter value for id: 03

old 1: insert into class values('&name',&id) new 1: insert into class values('chinthiya',03)

1 row created.

SQL> select * from class;

NAME	ID
anu	1
brindha	2
chinthiya	3

Create Sequence:

```
SQL> create sequence s_1
start with 4
increment by 1
maxvalue 100
cycle;
Sequence created.
SQL> insert into class values('divya',s_1.nextval);
1 row created.
SQL> select * from class;
```

NAME	ID
anu	1
brindha	2
chinthiya	3
divya	4

Alter Sequence:

```
SQL> alter sequence s_1  increment by 2;
Sequence altered.
SQL> insert into class values('fairoz',s_1.nextval);1 row created.
SQL> select * from class;
```

NAME	ID
Anu	1
brindha	2
chinthiya	3
divya	4
ezhil	5
fairoz	7

Drop Sequence:

```
SQL> drop sequence s_1;Sequence dropped.
```

RESULT:

Thus the views, synonyms and sequences were created and executed successfully.

Ex. No :05 **DATABASE PROGRAMMING: IMPLICIT AND EXPLICIT CURSORS**

Date :

AIM:

To create and implement implicit and explicit cursors in PL/SQL.

DECSRIPTION:

IMPLICIT CURSOR

PL/SQL declares an implicit cursor for every DML command, and queries that return a single row. The name ofthe implicit cursor is SQL. It can be used directly without any declaration.

EXPLICIT CURSOR

Explicit cursors are SELECT statements that are Declared explicitly in the declaration section of the current blockor in a package specification. Use OPEN, FETCH, and CLOSE in the execution or exception sections of the programs. The following example uses a cursor to select the five highest paid employees from the emps table.

SQL> create table student(id number, name varchar2(10), dept varchar2(10), percent number,m1 number,m2number, m3 number, tot number, g varchar2(1));

Table created.

SQL> select * from student;

ID	NAME	DEP	PERCENT	M1	M2	M3	TOT G
1	Anu	it	0	90	89	80	0
2	Beena	cse	0	98	91	95	0
3	Bindhu	it	0	87	67	86	0
4	Varun	it	0	67	46	50	0
5	Rahul	cse	0	81	82	83	0

SQL> declare

cursor c is select * from student;

ctot number;

cgra varchar2(1);

cper number;

begin 7 for I in c

loop

ctot= i.m1+i.m2+i.m3;

```
cper :=ctot/3;
update student set tot = ctot where id =i.id;
update student set percent = cper where id =i.id;
if(cper between 91 and 100)then
cgra:= ‘S’
elsif(cper between 81 and 90)then
cgra:= ‘A’
elsif(cper between 71 and 80)then
cgra:= ‘B’
elsif(cper between 61 and 70)then
cgra:= ‘C’
elsif(cper between 56 and 60)then
cgra:= ‘D’
elsif(cper between 50 and 55)then
cgra:= ‘E’
else
cgra:= ‘F’
end if;
update student set g = cgra where id =i.id;
end loop;
end;
31       /
```

PL/ SQL procedure successfully completed.SQL> select * from student;

ID	NAME	DEP	PERCENT	M1	M2	M3	TOT	G
1	Anu	it	86.3333333	90	89	80	259	A
2	Beena	cse	94.6666667	98	91	95	284	S
3	Bindhu	it	80	87	67	86	240	B
4	Varun	it	54.3333333	67	46	50	163	E
5	Rahul	cse	82	81	82	83	246	A

EXPLICIT CURSOR:

SYNTAX:

cursor cursor_name is select * from table name;To open the cursor:

open cursor_name;

To close the cursor:

close cursor_name;

Exercise:

Write PL/ SQL code for calculating hra , da, netsalary for all the employees in the Payroll Processing using Explicit cursor(uses employee table).

SQL> select * from employee;

EMPNO	NAME	HRA	DA	PF	NETSAL	BASICPAY
101	AAA	0	0	0	0	15000
102	BBB	0	0	0	0	18000
103	CCC	0	0	0	0	20000
104	DDD	0	0	0	0	10000
105	EEE	0	0	0	0	25000

```
SQL> declare
cursor c is select * from employee;
i employee% rowtype;
hrasal number;
dasal number;
pfsal number;
netsalary number;
begin
open c;
loop;
fetch c into i;
if c% notfound ten exit;
endif;
hrasal:=i.basicpay*0.1;
dasal:=i.basicpay*0.08;
pfsal:=i.basicpay*0.12;
netsalaray:= i.basicpay + hrasal + dasal + pfsal;
update employee set hra = hrasal, da= dasal, pf= pfsal, netsal= netsalaray where empno=i.empno;
end loop;
close c;
end;
```

PL/ SQL procedure successfully completed.SQL> select * from employee;

EMPNO	NAME	HRA	DA	PF	NETSAL	BASICPAY
101	AAA	1500	1200	1800	15900	15000
102	BBB	1800	1440	2160	19080	18000
103	CCC	2000	1600	2400	21200	20000
104	DDD	1000	800	1200	10600	10000
105	EEE	2500	2000	3000	26500	25000

RESULT:

Thus the implicit and explicit cursor have been created and executed successfully.

Ex.No.:06 **PROCEDURES AND FUNCTIONS**

Date :

AIM:

To create a function and procedure in PL/SQL and apply the same in SQL queries.

DESCRIPTION:

A procedure is a block that can take parameters (sometimes referred to as arguments) and be invoked. Procedures promote reusability and maintainability. Once validated, they can be used in number of applications.If the definition changes, only the procedure are affected, this greatly simplifies maintenance.

Modularized program development:

- Group logically related statements within blocks.
- Nest sub-blocks inside larger blocks to build powerful programs.

Break down a complex problem into a set of manageable well defined logical modules and implement the modules with blocks.

Procedure and function blocks:

Procedure:

- No return.
- PROCEDURE name IS
- Function:
- Returns a value
- FUNCTION name RETURN data-type IS

Syntax for procedure: Create [or Replace] PROCEDURE procedur_name(parameter1 [model1] datatype1,(parameter2 [model2] datatype2,…)

IS|AS

PL/SQL Block;

Syntax for function: Create [or Replace] function function_name (parameter1 [model1] datatype1, (parameter2 [model2] datatype2, …) return type

IS|AS

PL/SQL Block;

Example:

Create [or Replace] PROCEDURE leave_emp (v_id IN emp.empno%TYPE)IS BEGIN

DELETE from emp WHERE empno=v_id; END leave_emp;

SQL> create table stud(rno number(2),mark1 number(3),mark2 number(3),total number(3),primary key(rno));

Table created.

SQL> desc stud;

Name	Null?	Type
RNO	NOT NULL	NUMBER(2)
MARK1		NUMBER(3)
MARK2		NUMBER(3)
TOTAL		NUMBER(3)

SQL> select * from stud;

RNO	MARK1	MARK2	TOTAL
1	80	85	0
2	75	84	0
3	65	80	0
4	90	85	0

```
SQL> create or replace procedure stud (rnum number) is
m1 number;
m2 number;
total number;
begin
select mark1,mark2 into m1,m2 from stud where rno=rnum;
if m1<m2 then
update stud set total=m1+m2 where rno=rnum;
end if;
end;
11 /
```

Procedure created.

SQL> exec stud(1);

PL/SQL procedure successfully completed.

SQL> select * from stud;

```
RNO    MARK1        MARK2        TOTAL

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

       1      80          85         165
       2      75          84         0
       3      65          80         0
       4      90          85         0
SQL> exec studd(4);
PL/SQL procedure successfully completed.
SQL> select * from stud;
RNO    MARK1        MARK2        TOTAL

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

       1      80          85          165
       2      75          84          0
       3      65          80          0
       4      90          85          0
SQL> desc emp17;
Name          Null?          Type

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

ENO           NOT NULL     NUMBER(2)
ENAME         NOT NULL     VARCHAR2(18)
DNO           NOT NULL     NUMBER(3)
SAL                        NUMBER(8)
MID                        NUMBER(3)
SQL> select * from emp17;
         ENO   ENAME         DNO    SAL     MID

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

       1   Akshaya         102          50000       1
       2   Srikantan       105          12000       1
       3   Banupriya       100          32000       1
```

```
4   Chamundi      100        28000     3
5   Janani        101        24000     3
6   Subha         100        20000     4
7   Sridhar       105        35000     1
8   Shree         105        10000     2
9   Krithi        103        29000     3
9 rows selected.
SQL> create or replace procedure dnsal(enum number) is
s1 number;
sal number;
begin
select sal into s1 from emp17 where eno=enum;6 if s1>30000 then
 update emp17 set sal=s1+500 where eno=enum;8 end if;
if s1<30000 then
update emp17 set sal=s1+250 where eno=enum;11 end if;
 end;
 /
Procedure created.
SQL> exec dnsal(8);
PL/SQL procedure successfully completed.
SQL> select * from emp17 where eno=8;
        ENO ENAME  DNO    SAL     MID

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

        8        Shree  105     10250  2
SQL> exec dnsal(1);
PL/SQL procedure successfully completed.
SQL>  select * from emp17 where eno=1;
        ENO ENAME  DNO    SAL     MID

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

        1        Akshaya        102     50500  1
```

Function:

```
SQL> create table stud(rno number(2),mark1 number(3),mark2 number(3),total    number(3),primary
key(rno));
Table created.
SQL> desc stud;
         Name           Null?    Type

  ----------------------------------------------------
         RNO            NOT NULL     NUMBER(2)
         MARK1                       NUMBER(3)
         MARK2                       NUMBER(3)
         TOTAL                       NUMBER(3)
SQL> select * from stud;
RNO    MARK1          MARK2          TOTAL

       ----------------------------------
       1      80             85               0
       2      75             84               0
       3      65             80               0
       4      90             85               0
SQL> create or replace function stude(rnum number) return number is
total number;
m1 number;
m2 number;
begin
select mark1,mark2 into m1,m2 from stud where rno=rnum;
total:=m1+m2;
return total;
end;
10 /
Function created.
SQL> select stude(2) from dual;
         STUDE (2)

           -------
            159
SQL>  create  table  purchase(icode  number(3),  iname  varchar2(13),price  number(6),  quantity
number(3),rate number(8),primary key(icode),unique(iname));
Table Created
```

```
SQL>desc purchase;
        Name          Null?          Type
 ----------------------------------------------------

 ICODE                NOT NULL       NUMBER(3)
 INAME                               VARCHAR2(13)
 PRICE                               NUMBER(6)
 QUANTITY                            NUMBER(3)
 RATE                                NUMBER(8)
SQL> select * from purchase;
        ICODE NAME    PRICE   QUANTITY        RATE

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

      100     PenSet          20      10              0
      101     ParkerPen       60      10              0
      102     180pg Note      24      10              0
      103     80pg Note       10      25              0
      104     StickFile       10      20              0
SQL> create or replace function pur(itmcd number) return number is qt number;
pr number;
rate number;
begin
select price,quantity into pr,qt from purchase where icode=itmcd;
rate:=qt*pr;
return rate;
end;
10      /
Function created.
SQL> select pur(102) from dual;
    PUR(102)

   -------
        240
```

RESULT:

Thus the procedures and functions in PL/SQL were executed successfully.

Ex.No.: 07 **TRIGGERS**

Date :

AIM:

To create a trigger for insertion and deletion operation.

TRIGGER:

specify when a trigger is to be executed this is broken up into an event that cause the trigger to be checkedand as condition that must be satisfied for trigger execution is proceed.

Specify the action to be taken when trigger executes.

SYNTAX:

Create or replace trigger trigger-nameAfter/before

Insert/delete/update of columnnameOn tablename

For each row when conditionPL/SQL block;

PROGRAM:

TRIGGER WITH BEFORE UPDATETABLE

SQL> create table orders(order_id number(5),quantity number(4),cost_per_item number(6,2),total_cost number(8,2),updated_date date,updated_by varchar2(10));

Table created.

INSERT

SQL> insert into orders(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_item);

nter value for order_id: 1Enter value for quantity: 4

Enter value for cost_per_item: 20

old 1: insert into orders(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_it new 1: insert into orders(order_id,quantity,cost_per_item) values(1,4,20)

1 row created.

SQL> select * from orders;

ORDER_ID	QUANTITY	COST_PER_ITEM
1	4	20
2	5	30
3	6	25

TRIGGER

```
SQL> create or replace trigger orders_before_update before update on orders for each row declare
v_username varchar2(10);
begin
select user into v_username from dual;
new.updated_date:=sysdate;
new.updated_by:=v_username;
end;
12 /
Trigger created.

SQL> update orders set total_cost=3000 where order_id=2;
1 row updated.
SQL> select * from orders;
        ORDER_ID  QUANTITY  COST_PER_ITEM

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

       1                  20
       2                  30            3000
```

TRIGGER WITH AFTER UPDATE TABLE

```
SQL> create table orders30(order_id number(5),quantity number(4),cost_per_item number(6,2),total_cost
number(8,2));
Table created.
SQL> create table orders_audit(order_id number,quantity_before number,quantity_after number,
username varchar2(20));
Table created.
SQL> insert into orders30(order_id,quantity,cost_per_item)values(&order_id,&quantity,&cost_per_item);
Enter value for order_id: 100Enter value for quantity: 5
Enter value for cost_per_item: 10
old 1: insert into orders30(order_id,quantity,cost_per_item) values(&order_id,&quantity,&cost_per_ new 1:
insert into orders30(order_id,quantity,cost_per_item) values(100,5,10)
1 row created.
SQL> create or replace trigger orders_after_update
AFTER UPDATE
```

```
ON orders30
for each row declare
v_username varchar2(10);
begin
select user into v_username from dual;
insert into orders_audit
(order_id,quantity_before,quantity_after,username)
values
(:new.order_id,:old.quantity,:new.quantity,v_username);
end;
21 /
Trigger created.
SQL> update orders30 set quantity=25 where order_id=101;
1 row updated.
SQL> select *from orders_audit;

ORDER_ID QUANTITY_BEFORE QUANTITY_AFTER      USERNAME

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

101      4         25       CSE3090
```

RESULT:

Thus the program for trigger for insertion and deletion was successfully executed and output verified.

Ex.No.:07 **EXCEPTION HANDLING**

Date :

AIM:

To write a PL/SQL program with Exception handling mechanisms.

DESCRIPTION:

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exceptionHandling. Using Exception Handling we can test the code and avoid it from exiting abruptly.

When an exception occurs a messages which explains its cause is received. PL/SQL Exception message consists of three parts.

Type of Exception

An Error Code

A message

General Syntax for coding the exception section

DECLARE

Declaration sectionBEGIN

Exception sectionEXCEPTION

WHEN ex_name1 THEN

-Error handling statementsWHEN ex_name2 THEN

-Error handling statementsWHEN Others THEN

-Error handling statementsEND;

Program with user defined exception:

```
SQL> DECLARE
N INTEGER:=&N;
A EXCEPTION;
B EXCEPTION;
BEGIN
IF MOD(N,2)=0 THEN
RAISE A;
ELSE
RAISE B;
END IF;
EXCEPTION
WHEN A THEN
```

```
DBMS_OUTPUT.PUT_LINE('THE INPUT IS EVEN.')
WHEN B THEN
DBMS_OUTPUT.PUT_LINE('THE INPUT IS ODD.  ');
END;
 /
Enter value for n: 20
old 2: N INTEGER:=&N;new 2: N INTEGER:=20;
THE INPUT IS EVEN.....
PL/SQL procedure successfully completed.SQL> /
Enter value for n: 21
old 2: N INTEGER:=&N; new 2: N INTEGER:=21;THE INPUT IS ODD.....
PL/SQL procedure successfully completed.
```

Program with system defined exception:

Divide by zero exception:

```
SQL> DECLARE
L_NUM1 NUMBER;
L_NUM2 NUMBER;

 BEGIN
  L_NUM1 := 10;
  L_NUM2 := 0;
  DBMS_OUTPUT.PUT_LINE('RESULT:'||L_NUM1/L_NUM2);
EXCEPTION
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE(SQLCODE);
DBMS_OUTPUT.PUT_LINE(SQLERRM);
 END;
/

ORA-01476: divisor is equal to zero PL/SQL procedure successfully completed.
```

Handling the Exceptions on 'no data found'

```
SQL> create table employee1 (id number, employee_type_id ,number,external_id varchar2(30),
```

```
first_name      varchar2(30),
middle_name  varchar2(30),
last_name      varchar2(30),
name   varchar2(100),
birth_date      date ,
gender_id       number );
Table created.
SQL> create table gender ( id number, codevarchar2(30), description varchar2(80), active_date date
          default SYSDATE not null, inactive_date date );
Table created.
SQL> insert into gender ( id, code, description ) values ( 1, 'F', 'Female' );1 row created.
SQL> insert into gender ( id, code, description ) values ( 2, 'M', 'Male' );1 row created.
SQL> insert into gender ( id, code, description ) values ( 3, 'U', 'Unknown' );1 row created.
SQL> set serveroutput on size 1000000;SQL> declare

        d_birth_date          employee1.birth_date%TYPE;
        n_gender_id           employee1.gender_id%TYPE;
        n_selected            number := -1;
        n_id                  employee1.id%TYPE;
        v_first_name          employee1.first_name%TYPE;
        v_last_name           employee1.last_name%TYPE;
        v_middle_name         employee1.middle_name%TYPE;
       v_name   employee1.name%TYPE;11
begin
v_first_name := 'JOHN';
v_middle_name := 'J.';
v_last_name  := 'DOUGH';
v_name          := rtrim(v_last_name||', '||v_first_name||' '||v_middle_name);
d_birth_date := to_date('19800101', 'YYYYMMDD');18
begin
select id into n_gender_id from gender where code = 'M';
exception
when OTHERS then
raise_application_error(-20001, SQLERRM||' on select gender');
end;
```

```
begin
select id
into   n_id
from   employee1
where  name   = v_name
and      birth_date = d_birth_date
and      gender_id = n_gender_id;33
n_selected := sql%rowcount;
exception
when NO_DATA_FOUND then
n_selected := sql%rowcount;
DBMS_OUTPUT.PUT_LINE('Caught raised exception NO_DATA_FOUND');
when OTHERS then
raise_application_error(-20002, SQLERRM||' on select employee');
end;

DBMS_OUTPUT.PUT_LINE(to_char(n_selected)||' row(s) selected.');
end;
 /
Caught raised exception NO_DATA_FOUND0 row(s) selected.
PL/SQL procedure successfully completed.
```

RESULT:

Thus the PL/SQL for exception handling has been created and executed successfully.

Ex.No.:09 DATABASE DESIGN USING ER MODELING, NORMALIZATION AND DATE
Date: IMPLEMENTATION FOR ANY APPLICATION

AIM:

To create a database application using ER modeling and perform normalization in it.

Description:

Roadway Travels SystemRequirements Analysis Roadway Travels:

Roadway travels is in business since 1997 with several buses connecting different places in india. Its main office is located in Hyderabad. The company wants to computerize its operations in the following areas.

The company wants to computerize its operations in the following areas:

- Reservations and Ticketing
- Cancellations

Reservations & Cancellation:

Reservations are directly handled by booking office. Reservations can be made 30 days in advance and tickets issued to passenger. One passenger/ person can book many tickets (to his/her family).Cancellations arealso directly handed at the booking office.

In the process of Computerization of Roadway Travels you have to design and develop a Database which consists the data of Buses, Passengers, Tickets and Reservation and cancellation details. You should also develop query's using SQL to retrieve the data from the database.

Following steps are involved in the process:

Analyzing the problem and identifying the Entities and Relationships

E-R Model

Normalised Relational Model

Creating the database

Querying.

Triggers and Stored procedures on the tables.

1) E-R Model:

Analyze the problem carefully and come up with the entities in it. Identify what data has to be persisted in the database. This contains the entities, attributes etc. In this we wqill analyze different types of entities with attributes of "Roadways Travels".

Entity: An Entity is an object or concept about which you want to store information

Relationship: A relationship is an association between several entities.

Attributes: An object is characterized by its properties or attributes. In relational database systems

attributescorrespond to fields.

The Road Way Travels Consists Of The Following Entities:

BUS

Ticket Passenger Reservation

Cancellation/modification

These Entities have following attributes

Bus:

Bus_id Bus_name Bus_type Bus_totalseats

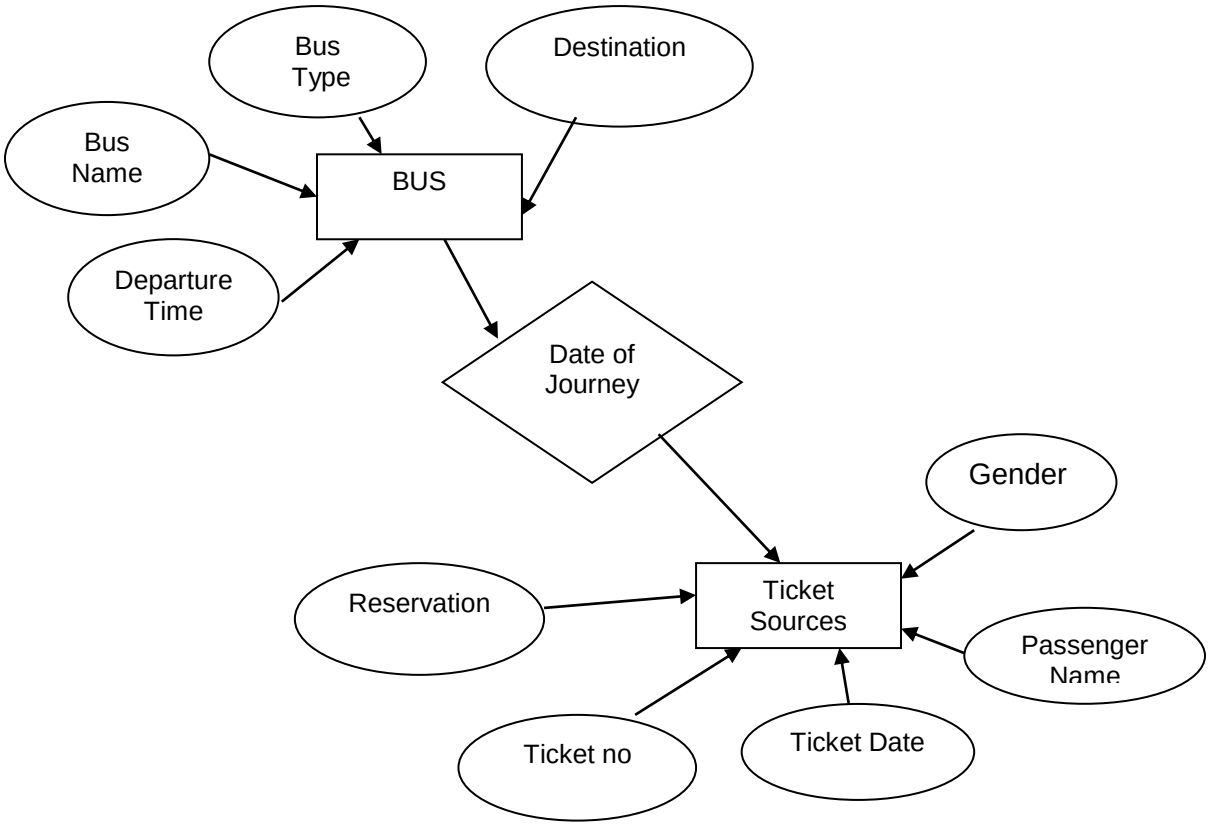
Ticket:

Ticket_booking Ticket_datejourneyTicket_to Ticket_from Ticket_id Ticket_no of tickets

Passenger:

Pid Pname Pgender Page precancel

E_R diagram:



Normalization

Database normalization is a technique for designing relational database tables to minimize duplication of information and, in so doing, to safeguard the database against certain types of logical or structural problems, namely data anomalies. In this we will write the normalization tables that is entities of "Roadway Travels."

The objectives of normalization beyond 1NF (First Normal Form) were stated as follows by Codd:

- To free the collection of relations from undesirable insertion, update and deletion dependencies;
- To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thusincrease the life span of application programs;
- To make the relational model more informative to users;
- To make the collection of relations neutral to the query statistics, where these statistics are liable to change astime goes by.

Querying and manipulating the data within a data structure which is not normalized, such as the following non-1NF representation of customers' credit card transactions, involves more complexity than is really necessary:

Customer		**Transactions**	
	Tr.ID	**Date**	**Amount**
Jones	12890	14-OCT-2003	87
	12904	15-OCT-2003	50
	Tr.ID	**Date**	**Amount**
Wilkinson	12898	14-OCT-2003	21
	Tr.ID	**Date**	**Amount**
Stevens	12907	15-OCT-2003	18
	14920	20-NOV-2003	70
	15003	27-NOV-2003	60

To each customer corresponds a repeating group of transactions. The automated evaluation of any query relatingto customers' transactions therefore would broadly involve two stages:

Unpacking one or more customers' groups of transactions allowing the individual transactions in a group to be examined, and

Deriving a query result based on the results of the first stage

For example, in order to find out the monetary sum of all transactions that occurred in October 2003 for all

customers, the system would have to know that it must first unpack the *Transactions* group of each customer, thensum the *Amounts* of all transactions thus obtained where the *Date* of the transaction falls in October 2003.

One of Codd's important insights was that this structural complexity could always be removed completely,leading to much greater power and flexibility in the way queries could be formulated (by users and applications) and evaluated (by the DBMS). The normalized equivalent of the structure above would look like this

Customer	**Tr. ID**	**Date**	**Amount**
Jones	12890	14-Oct-2003	−87
Jones	12904	15-Oct-2003	−50
Wilkins	12898	14-Oct-2003	−21
Stevens	12907	15-Oct-2003	−18
Stevens	14920	20-Nov-2003	−70
Stevens	15003	27-Nov-2003	−60

RESULT:

Thus the database application using ER modeling has been created and executed successfully.

Ex.No.:10 **DATABASE CONNECTIVITY WITH FRONT END TOOLS**

Date :

AIM:

To connect Oracle database to Visual Basic using ODBC (Open DataBase Connectivity).

STEPS:

Step-1: Design database schema using Oracle DadaBase.

Step-2: Create student information system with attributes student_id,student_name, age and department.

Step-3: Insert the values for the schema and save (commit).

Step-4: Perform the following setps for ODBC connectivity goto ->settings ->control panel -> performance and maintenances ->Administrative tools->Data sources(ODBC).

Step-5: Select User DSN -> ADD

Step-6: Select a driver for ORACLE ->finish

Step-7: Set Data source name(DSN) as SIS, description Student Information System ,User Id(UID) as scott.

Then press OK and test the connection.

Step-8: Design a GUI application to accept student details from the Student Information System schema andwrite query for searching student details based on student_id

PROGRAM:

Database Design:

SQL> create table sis(std_id varchar2(10) primary key,std_name varchar2(10),

age number(3), dept varchar2(5)); Table created.

SQL> insert into sis values('&std_id','&name',&age,'&dept');Enter value for std_id: 1

Enter value for name: saranEnter value for age: 30 Enter value for dept: cse

old 1: insert into sis values('&std_id','&name',&age,'&dept')new 1: insert into sis values('1','saran',30,'cse')

1 row created.

SQL> /

Enter value for std_id: 2 Enter value for name: ramEnter value for age: 22 Enter value for dept: it

old 1: insert into sis values('&std_id','&name',&age,'&dept')new 1: insert into sis values('2','ram',22,'it')

1 row created.

SQL> /

Enter value for std_id: 3 Enter value for name: syamEnter value for age: 19 Enter value for dept: ece

old 1: insert into sis values('&std_id','&name',&age,'&dept')new 1: insert into sis values('3','syam',19,'ece')

1 row created.

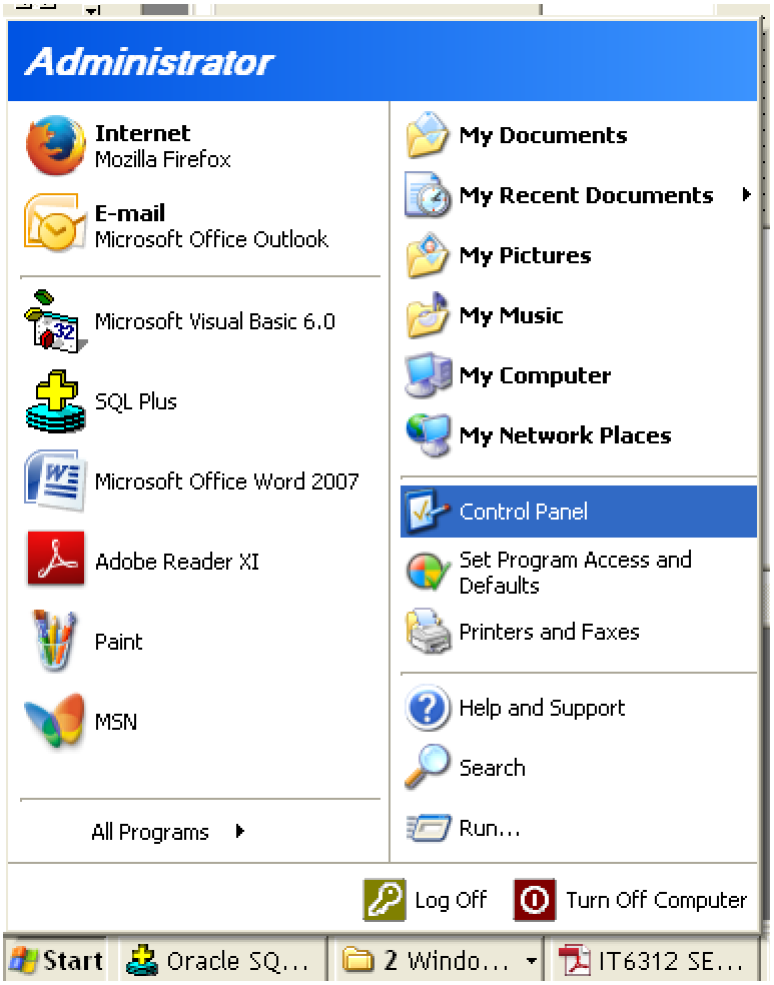
SQL> select * from sis;

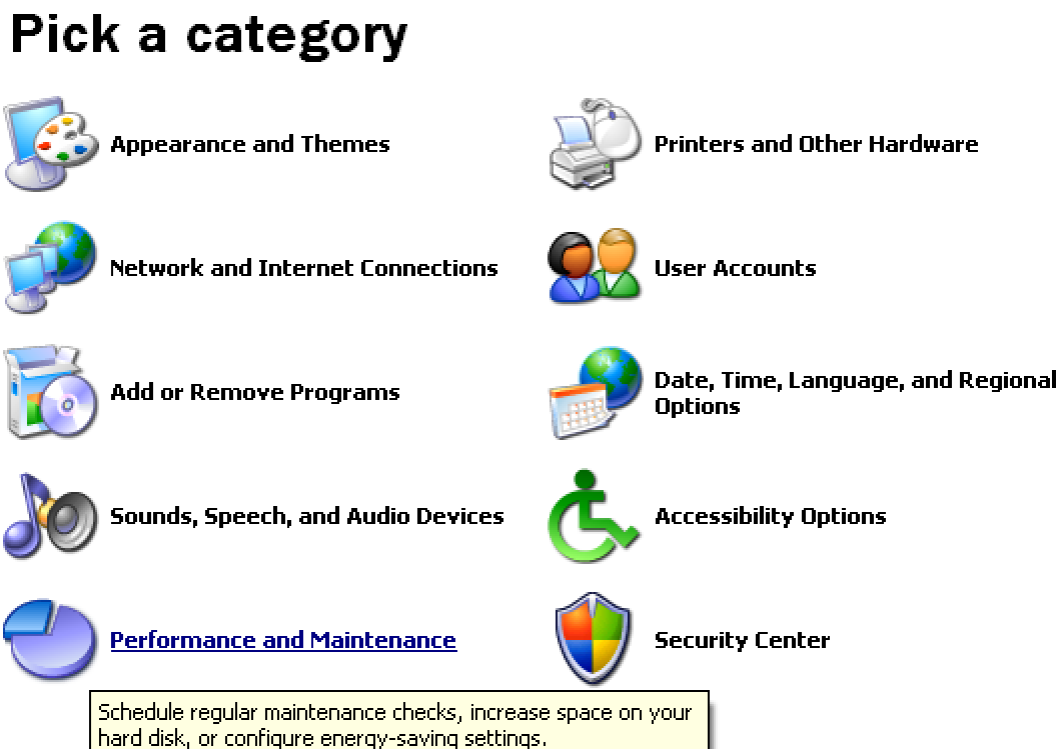
STD_ID	STD_NAME	AGE	DEPT
1	saran	30	cse
2	ram	22	it
3	syam	19	ece

SQL> commit;

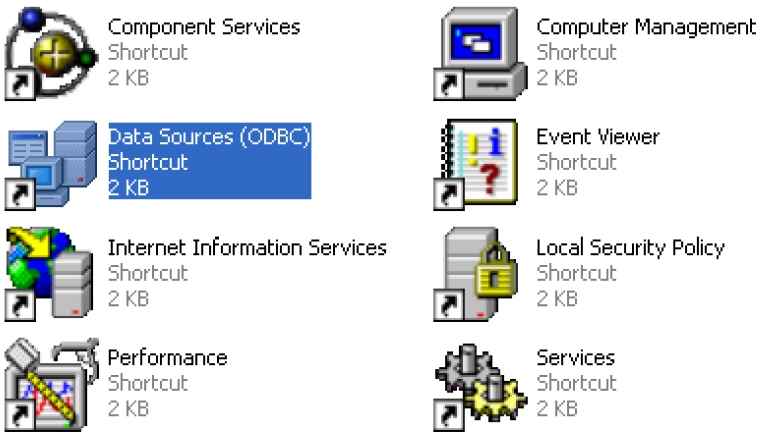
Commit complete.

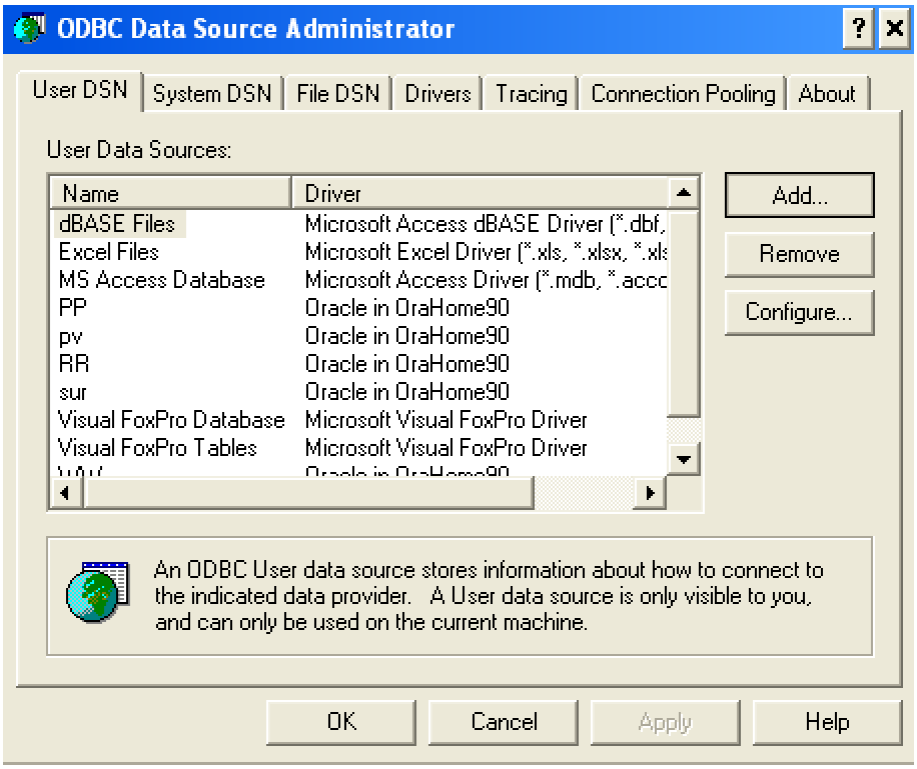
ODBC CONNECTIVITY:

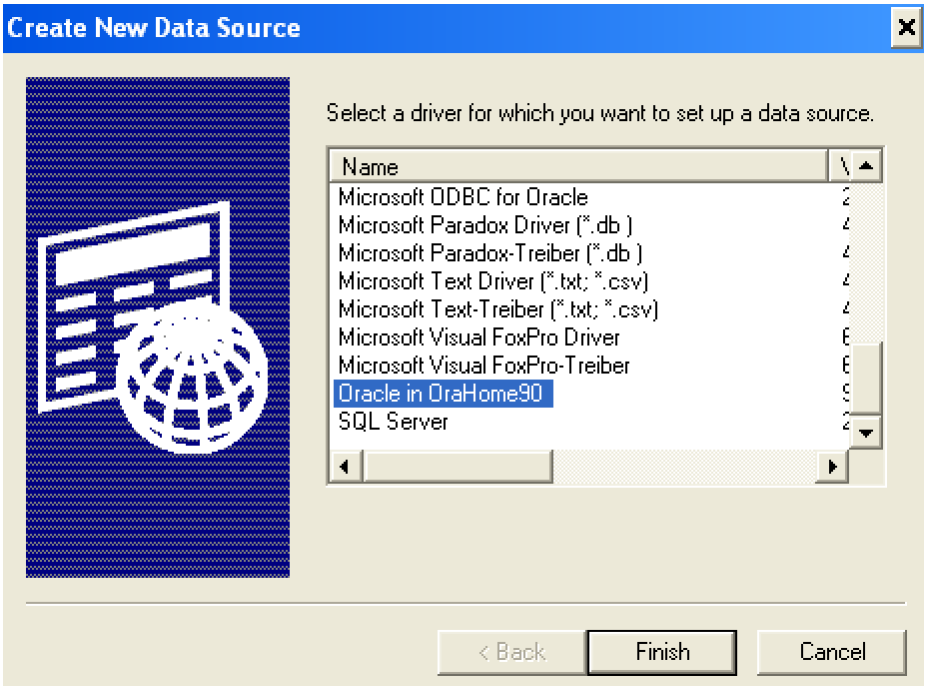


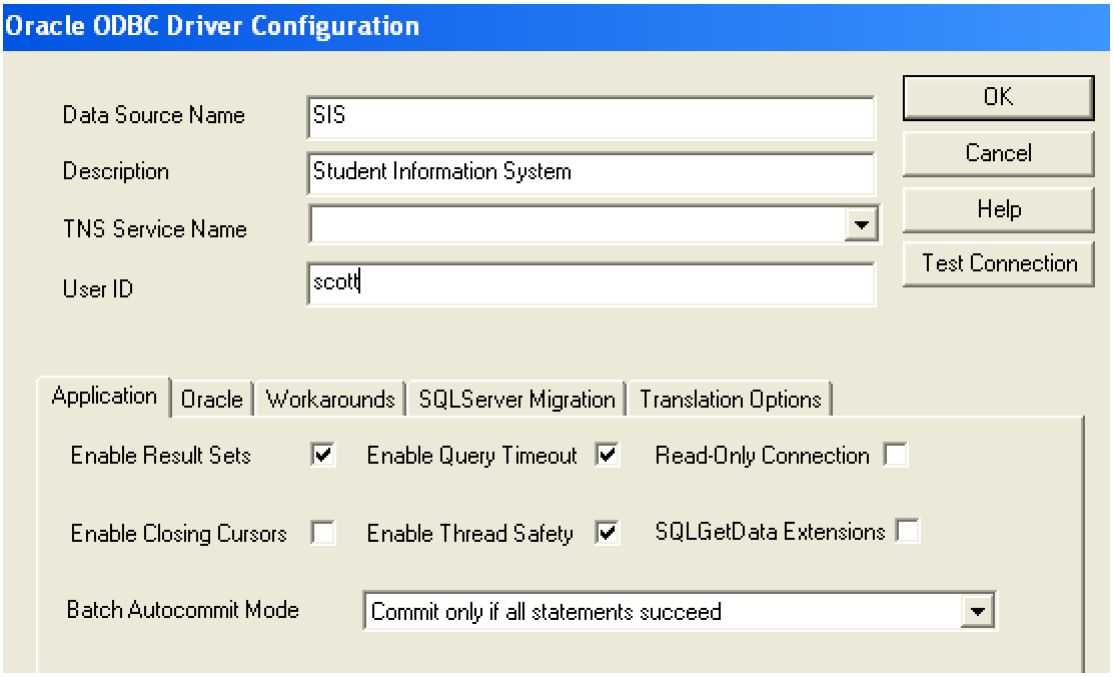

Pick a category
Appearance and Themes
Printers and Other Hardware
Network and Internet Connections
User Accounts
Add or Remove Programs
Date, Time, Language, and Regional Options
Sounds, Speech, and Audio Devices
Accessibility Options
Performance and Maintenance
Security Center
Schedule regular maintenance checks, increase space on your hard disk, or configure energy-saving settings.

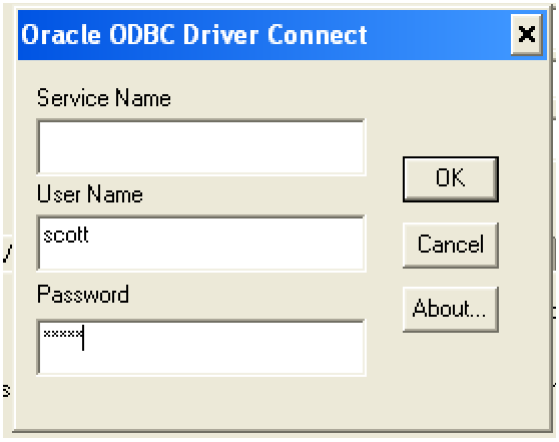

or pick a Control Panel icon
Administrative Tools
Power Options
Configure administrative settings for your computer.
Scheduled Tasks
System

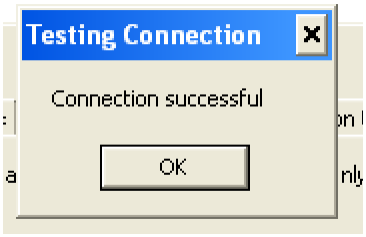

Component Services
Shortcut
2 KB
Computer Management
Shortcut
2 KB
Data Sources (ODBC)
Shortcut
2 KB
Event Viewer
Shortcut
2 KB
Internet Information Services
Shortcut
2 KB
Local Security Policy
Shortcut
2 KB
Performance
Shortcut
2 KB
Services
Shortcut
2 KB


ODBC Data Source Administrator
User DSN
System DSN
File DSN
Drivers
Tracing
Connection Pooling
About
User Data Sources:
Name
Driver
dBASE Files
Microsoft Access dBASE Driver (*.dbf,
Excel Files
Microsoft Excel Driver (*.xls, *.xlsx, *.xls
MS Access Database
Microsoft Access Driver (*.mdb, *.accc
PP
Oracle in OraHome90
pv
Oracle in OraHome90
RR
Oracle in OraHome90
sur
Oracle in OraHome90
Visual FoxPro Database
Microsoft Visual FoxPro Driver
Visual FoxPro Tables
Microsoft Visual FoxPro Driver
Add...
Remove
Configure...
An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.
OK
Cancel
Apply
Help


Create New Data Source
Select a driver for which you want to set up a data source.
Name
Microsoft ODBC for Oracle
Microsoft Paradox Driver (*.db)
Microsoft Paradox-Treiber (*.db)
Microsoft Text Driver (*.txt; *.csv)
Microsoft Text-Treiber (*.txt; *.csv)
Microsoft Visual FoxPro Driver
Microsoft Visual FoxPro-Treiber
Oracle in OraHome90
SQL Server
< Back
Finish
Cancel

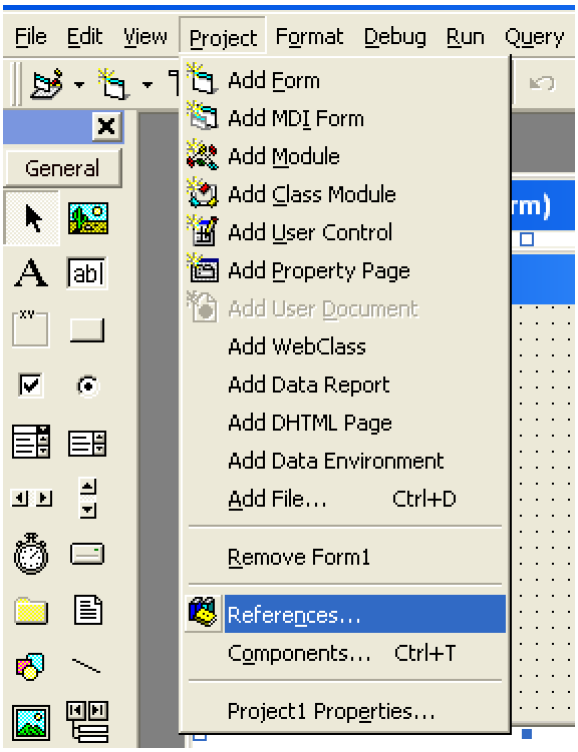

Oracle ODBC Driver Configuration
Data Source Name
SIS
Description
Student Information System
TNS Service Name
User ID
scott
OK
Cancel
Help
Test Connection
Application
Oracle
Workarounds
SQLServer Migration
Translation Options
Enable Result Sets
Enable Query Timeout
Read-Only Connection
Enable Closing Cursors
Enable Thread Safety
SQLGetData Extensions
Batch Autocommit Mode
Commit only if all statements succeed

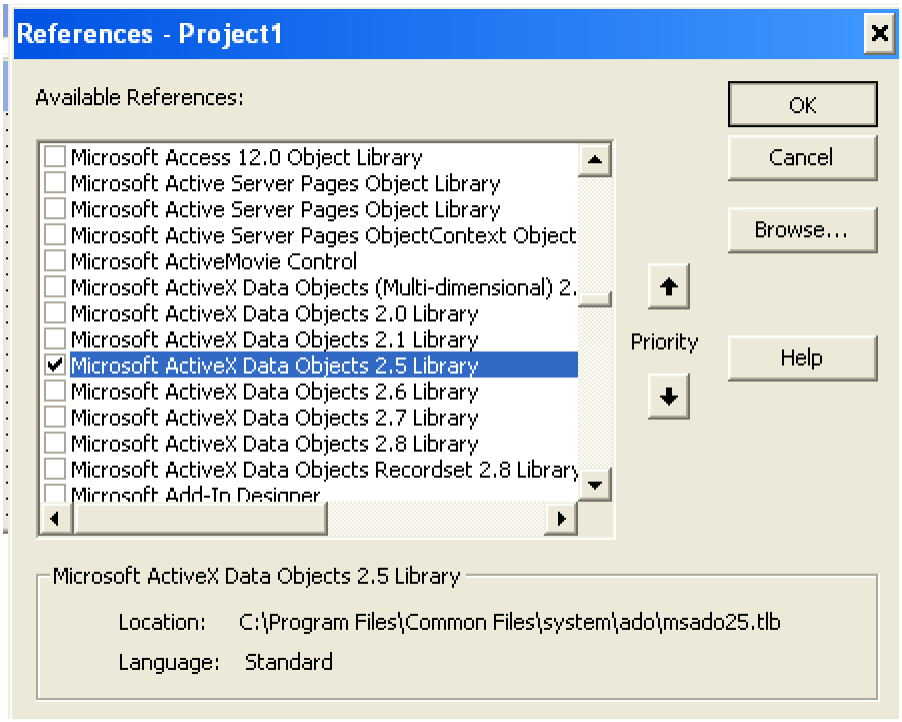

Oracle ODBC Driver Connect
Service Name
User Name
scott
Password
OK
Cancel
About...

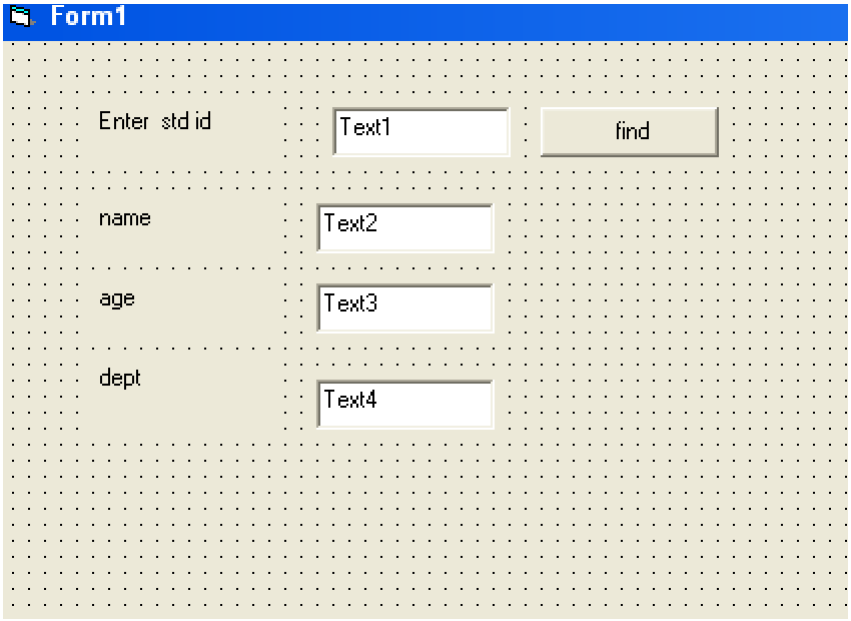

Testing Connection
Connection successful
OK

GUI APPLICATION FOR DATABASE SCHEMA –SIS (USING VISUAL BASIC 6.0)

FORM DESIGN:







CODING:

Dim rs As New ADODB.RecordsetPrivate Sub Command1_Click()

Dim ss = ""

Do Until rs.EOF = True

If rs(0) = Text1.Text ThenText2.Text = rs(1) Text3.Text = rs(2) Text4.Text = rs(3)

s = 1 Exit Dors.Close Else

rs.MoveNextEnd If

Loop

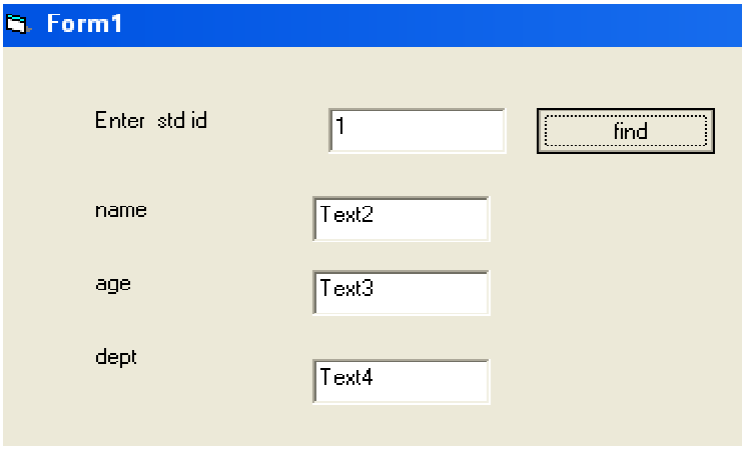
If s = "" Then MsgBox "not exist"End If

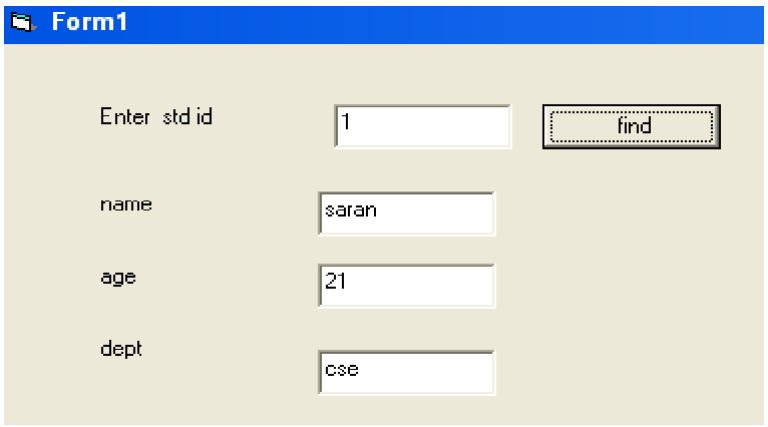
End Sub

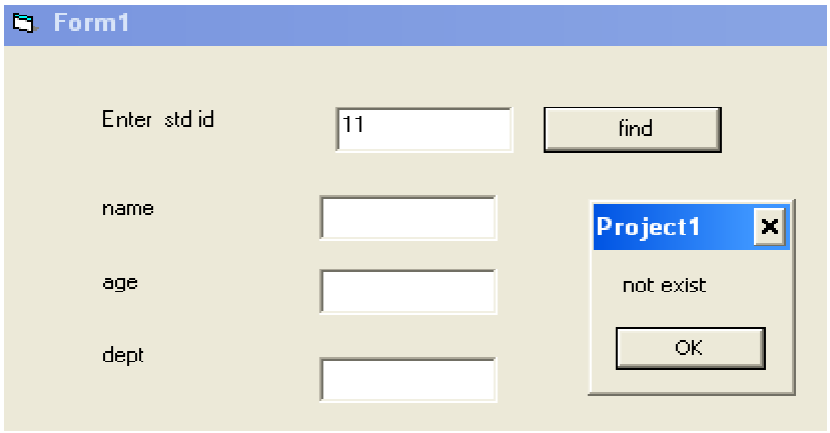
Private Sub Form_Load()

rs.Open "select * from sis;", "dsn=sis;uid=system;pwd=manager;", adOpenDynamic, adLockOptimisticEnd Sub

OUTPUT:







RESULT:

Thus the ODBC connectivity has been successfully established.

Ex.No.:11 Case Study Using Real Life Database Application

Date :

AIM:

To study about real life database application.

DESCRIPTION:

The case study teaching method is a highly adaptable style of teaching that involves problem-based learning and promotes the development of analytical skills. By presenting content in the format of a narrative accompanied by questions and activities that promote group discussion and solving of complex problems, case studies facilitate development of the higher levels of Bloom's taxonomy of cognitive learning; moving beyond recall of knowledge to analysis, evaluation, and application . Similarly, case studies facilitate interdisciplinary learning and can be used to highlight connections between specific academic topics and real-world societal issues and applications. This has been reported to increase student motivation to participate in class activities, which promotes learning and increases performance on assessments. For these reasons, case-based teaching has been widely used in business and medical education for many years. Although case studies were considered a novel method of science education just 20 years ago, the case study teaching method has gained popularity in recent years among an array of scientific disciplines such as biology, chemistry, nursing, and psychology.

METHOD:

Student population

This study was conducted at Kingsborough Community College, which is part of the City University of New York system, located in Brooklyn, New York. Kingsborough Community College has a diverse population of approximately 19,000 undergraduate students. The student population included in this study was enrolled in the first semester of a two-semester sequence of general (introductory) biology for biology majors during the spring, winter, or summer semester of 2014. A total of 63 students completed the course during this time period; 56 students consented to the inclusion of their data in the study. Of the students included in the study, 23 (41%) were male and 33 (59%) were female; 40 (71%) were registered as college freshmen and 16 (29%) were registered as college sophomores. To normalize participant groups, the same student population pooled from three classes taught by the same instructor was used to assess both experimental and control teaching methods.

Course material

The four biological concepts assessed during this study (chemical bonds, osmosis and diffusion, mitosis and meiosis, and DNA structure and replication) were selected as topics for studying the effectiveness of

case study teaching because they were the key concepts addressed by this particular course that were most likely to be taught in a number of other courses, including biology courses for both majors and nonmajors at outside institutions. At the start of this study, relevant existing case studies were freely available from the National Center for Case Study Teaching in Science (NCCSTS) to address mitosis and meiosis and DNA structure and replication,

but published case studies that appropriately addressed chemical bonds and osmosis and diffusion were not available. Therefore, original case studies that addressed the latter two topics were produced as part of this study, and case studies produced by unaffiliated instructors and published by the NCCSTS were used to address the former two topics. By the conclusion of this study, all four case studies had been peer-reviewed and accepted for publication by the NCCSTS. Four of the remaining core topics covered in this course (macromolecules, photosynthesis, genetic inheritance, and translation) were selected as control lessons to provide control assessment data.

To minimize extraneous variation, control topics and assessments were carefully matched in complexity, format, and number with case studies, and an equal amount of class time was allocated for each case study and the corresponding control lesson. Instruction related to control lessons was delivered using minimal slide-based lectures, with emphasis on textbook reading assignments accompanied by worksheets completed by students in and out of the classroom, and small and large group discussion of key points. Completion of activities and discussion related to all case studies and control topics that were analyzed was conducted in the classroom, with the exception of the take-home portion of the osmosis and diffusion case study.

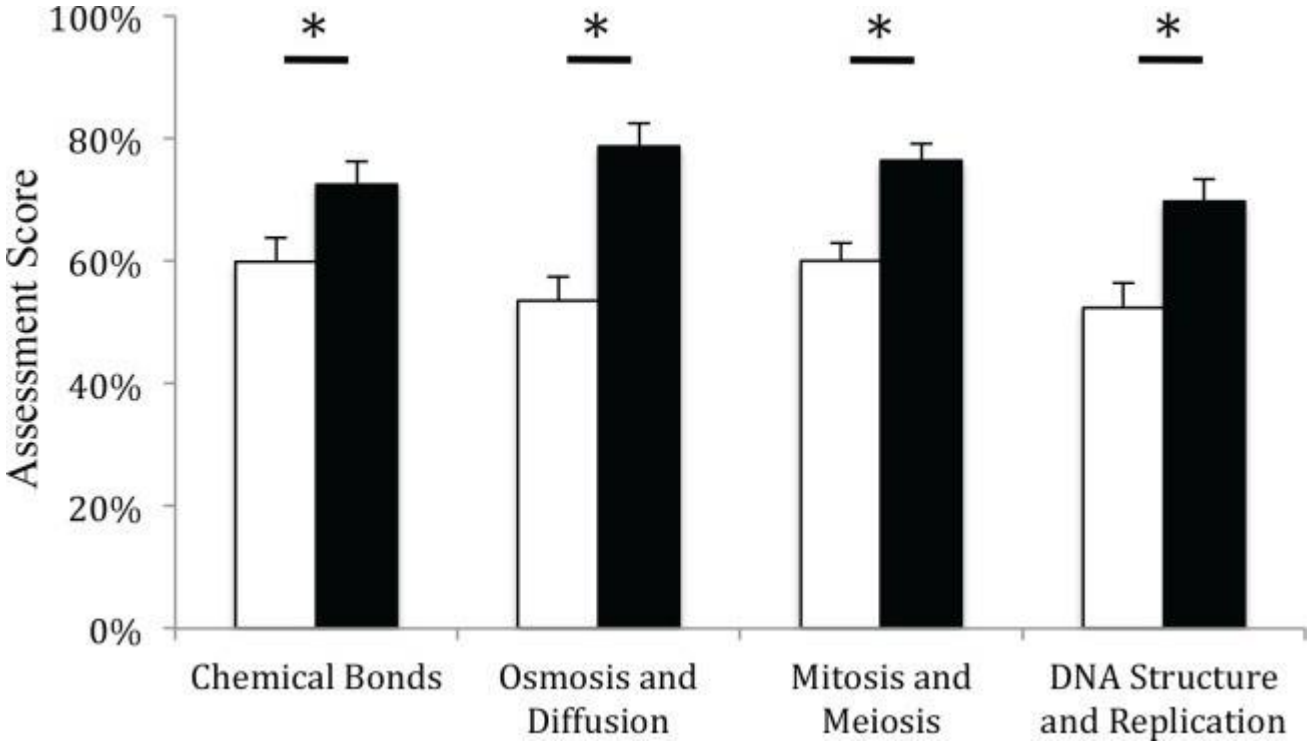
Data collection and analysis

This study was performed in accordance with a protocol approved by the Kingsborough Community College Human Research Protection Program and the Institutional Review Board (IRB) of the City University of New York (CUNY IRB reference 539938-1; KCC IRB application #: KCC 13-12-126-0138). Assessment scores were collected from regularly scheduled course examinations. For each case study, control questions were included on the same examination that were similar in number, format, point value, and difficulty level, but related to a different topic covered in the course that was of similar complexity. Complexity and difficulty of both case study and control questions were evaluated using experiential data from previous iterations of the course; the Bloom's taxonomy designation and amount of material covered by each question, as well as the average score on similar questions achieved by students in previous iterations of the course was considered in determining appropriate controls. All assessment questions were scored using a standardized, pre-determined rubric. Student perceptions of learning gains were assessed using a modified version of the Student Assessment of Learning Gains (SALG) course evaluation tool, distributed in hardcopy and completed anonymously during the last week of the course. Students were presented with a consent form to opt-in to having their data included in the

data analysis. After the course had concluded and final course grades had been posted, data from consenting students were pooled in a database and identifying information was removed prior to analysis. Statistical analysis of data was conducted using the Kruskal-Wallis one-way analysis of variance and calculation of the R^2 coefficient of determination.

Teaching with case studies improves performance on learning assessments, independent of case study origin .

To evaluate the effectiveness of the case study teaching method at promoting learning, student performance on examination questions related to material covered by case studies was compared with performance on questions that covered material addressed through classroom discussions and textbook reading. The latter questions served as control items; assessment items for each case study were compared with control items that were of similar format, difficulty, and point value. Each of the four case studies resulted in an increase in examination performance compared with control questions that was statistically significant, with an average difference of 18%. The mean score on case study-related questions was 73% for the chemical bonds case study, 79% for osmosis and diffusion, 76% for mitosis and meiosis, and 70% for DNA structure and replication. The mean score for non-case study-related control questions was 60%, 54%, 60%, and 52%, respectively. In terms of examination performance, no significant difference between case studies produced by the instructor of the course (chemical bonds and osmosis and diffusion) and those produced by unaffiliated instructors (mitosis and meiosis and DNA structure and replication) was indicated by the Kruskal-Wallis one-way analysis of variance. However, the 25% difference between the mean score on questions related to the osmosis and diffusion case study and the mean score on the paired control questions was notably higher than the 13–18% differences observed for the othercase studies.



Case study teaching method increases student performance on examination questions. Mean score on a set of examination questions related to lessons covered by case studies (black bars) and paired control questions of similar format and difficulty about an unrelated topic (white bars). Chemical bonds, n = 54; Osmosis and diffusion, n = 54; Mitosis and meiosis, n = 51; DNA structure and replication, n = 50. Error bars represent the standard error of the mean (SEM). Asterisk indicates $p < 0.05$.

RESULT:

Thus the real life database application has been studied.