LIST OF EXPERIMENTS

1. Data Definition Language Commands
2. Data Manipulation Language Commands
3. Integrity Constraints
4. Views
5. Data Control Language Commands
6. Transaction Control Language Commands
7. Simple programs using PL/SQL
8. Triggers
9. Goto and Exception handling
10. Implicit and Explicit Cursors
11. Procedures and functions
12. Embedded SQL
13. Application Development using appropriate Front End and Back End Tools for Employee Management System.
14. Construct a Distributed Data Base for BOOK STORE
15. MongoDB Client Setup, Installation
    a) Getting / Selecting Collection

DATA DEFINITION LANGUAGE COMMANDS

EX NO: 01
DATE:

AIM
To study the various DDL commands and implement them on the database.

COMMANDS

```
SQL> create table stud (sname varchar2(30), sid varchar2(10), sage number(2), sarea
varchar2(20));

Table created.

SQL> desc stud;
 Name Null? Type
 ------------------------------------------------- -------- ------------------------------
 SNAME VARCHAR2(30)
 SID VARCHAR2(10)
 SAGE NUMBER(2)
 SAREA VARCHAR2(20)

SQL>alter table stud modify ( sage number(10));
```

```
Table altered.


SQL> alter table stud add ( sdept varchar2(20));

Table altered.

SQL> desc stud;
 Name Null? Type
 -------------------------------------------------- -------- -------------------------------
 SNAME VARCHAR2(30)
 SID VARCHAR2(10)
 SAGE NUMBER(10)
 SAREA VARCHAR2(20)
 SDEPT VARCHAR2(20)

SQL> alter table stud drop ( sdept varchar2(20));

Table altered.
SQL> desc studs;
 Name Null? Type
 -------------------------------------------------- --------
 ----------------------------------- SNAME VARCHAR2(30)
 SID VARCHAR2(10)
 SAGE NUMBER(10)
 SAREA VARCHAR2(20)

SQL> truncate table studs;

Table truncated.

SQL> desc studs;
 Name Null? Type
 -------------------------------------------------- --------
 ----------------------------------- SNAME VARCHAR2(30)
 SID VARCHAR2(10)
 SAGE NUMBER(10)
 SAREA VARCHAR2(20)
 SDEPT VARCHAR2(20)

SQL> drop table studs;

Table dropped.
```

RESULT

Thus the DDL commands were implemented and the output was verified.

DATA MANIPULATION LANGUAGE COMMANDS

EX NO:02
DATE:

AIM

To study the various categories of DML commands such as logical operations, aggregate functions, string functions, numeric functions, date functions, conversion functions and group functions, set operations, join operations and nested queries..

DESCRIPTION

THE ORACLE TABLE – DUAL

Dual is a small oracle table which consists of only one row and one column and contains the value X in that column.

INSERT

This command is used to insert values into the table.

SELECT

This command is used to display the contents of the table or those of a particular column.

RENAME

This command renames the name of the table.

ARITHMETIC OPERATIONS

Various operations such as addition, multiplication, subtraction and division can be performed using the numbers available in the table.

DISTINCT

This keyword is used along with select keyword to display unique values from the specified column. It avoids duplicates during display.

ORDER BY CLAUSE

The order by clause arranges the contents of the table in ascending order (by default) or in descending order (if specified explicitly) according to the specified column.

CONCATENATION OPERATOR

This combines information from two or more columns in a sentence according to the format specified.

LOGICAL OPERATORS

- AND : The oracle engine will process all rows in a table and displays the result only when all of the conditions specified using the AND operator are specified. OR : The oracle engine will process all rows in a table and displays the result only when any of the conditions specified using the OR operators are satisfied. NOT : The oracle engine will process all rows in a table and displays the result only when none of the conditions specified using the NOT operator are specified.

 BETWEEN : In order to select data that is within a range of values, the between operator is used. (AND should be included)

PATTERN MATCH

 LIKE PREDICATE : The use of like predicate is that it allows the comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters which are % and _. The purpose of % is that it matches any string and _ matches any single character.

 IN AND NOT IN PREDICATE : The arithmetic operator = compares a single value to another single value. In case a value needs to be compared to a list of values then the in predicate is used.The not in predicate is the opposite of the in predicate. This will select all the rows whose values do not match all of the values in the list.

NUMERIC FUNCTIONS

 ABS: It returns the absolute value of 'n'.

 POWER: It returns m raised to nth power. n must be an integer else an error is returned.

 ROUND: It returns n rounded to m places right of the decimal point. If m is omitted, n is rounded to zero places. m must be an integer.

 SQRT: It returns square root of n. n should be greater than zero.

STRING FUNCTIONS

 LOWER: It returns char with letters in lower case.

 INITCAP: It returns char with the first letter in upper case.

 UPPER: It returns char with all letters forced to upper case.

 SUBSTR: It returns a portion of char beginning at character m, exceeding up to n characters. If n is omitted result is written up to the end character. The 1st position of char is one.

 LENGTH: It returns the length of char

 LTRIM: It removes characters from the left of char with initial characters removed up to the 1st character not in set.

 RTRIM: It returns char with final characters removed after the last character not in the set. Set is optional. It defaults to spaces.

 LPAD: It returns char1, left padded to length n with the sequence of characters in char2. char2 defaults to blanks.

 RPAD: It returns char1, right padded to length n with the characters in char2, replicated as many times as necessary. If char2 is omitted, it is padded with blanks.

AGGREGATE FUNCTIONS

 AVG (N): It returns average value of n ignoring null values.

 MIN (EXPR): It returns minimum value of the expression.

 COUNT (EXPR): It returns the number of rows where expression is not null. COUNT (*): It returns the number of rows in the table including the duplicates and those with null values.

 MAX (EXPR): It returns maximum value of the expression.

 SUM(N): It returns sum of values of n.

CONVERSION FUCTIONS

 TO_NUMBER(CHAR): It converts the char value containing a number to a value of number data type.

 TO_CHAR(N,FMT): It converts a value of number data type to a value of char data type, using the optional format string. It accepts a number n and a numeric format fmt in which the number has to appear. If fmt is omitted, n is converted to a char value exactly long enough to hold significant digits.

 TO_CHAR(DATE, FMT): It converts a value of data type to char value. It accepts a date as well as the format in which the date has to appear. Fmt must be a date format. If fmt is omitted, date is the default date format.

DATE FUNCTIONS

 SYSDATE : The sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table dual and returns the current date.

 ADD_MONTHS(D,N): It returns date after adding the number of months specified with the function.

 LAST_DAY(D): It returns the last date of the month specified with the function

 MONTHS_BETWEEN(D1,D2): It returns number of months between D1 and D2.

 NEXT_DAY(DATE, CHAR): It returns the date of the first week day named by char . char must be a day of the week.

GROUP BY CLAUSE

The group by clause is another section of the select statement. This optional class tells oracle to group rows based on distinct values that exists for specified columns.

HAVING CLAUSE

The having clause can be used in conjunction with the group by clause. Having imposes a condition on the group by clause, which further filters the groups created by the group by clause.

SET OPERATIONS

 UNION CLAUSE: Multiple queries can be put together and their output combined using the union clause. The union clause merges the output of two or more queries into a single set of rows and columns.

 INTERSECT CLAUSE: Multiple queries can be put together and their output can be combined using the intersect clause. The intersect clause outputs only rows produced by both the queries intersected. The output in an intersect clause will include only those rows that are retrieved by both the queries. **JOIN OPERATIONS**

 INNER JOIN/ NATURAL JOIN/ JOIN: It is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. OUTER JOIN: It is an extension of join operation to deal with missing information.

Left Outer Join: It takes tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join.

Right Outer Join: It takes tuples in the right relation that did not match with any tuple in the left relation, pads the tuples with null values for all other attributes from the left relation and adds them to the result of the natural join.

Full Outer Join: It combines tuples from both the left and the right relation and pads the tuples with null values for the missing attributes and them to the result of the natural join.

<u>**COMMANDS**</u>

CREATION OF TABLE

SQL>create table stud (sname varchar2(30), sid varchar2(10), sage number(10), sarea varchar2(20), sdept varchar2(20));

Table created.

INSERTION OF VALUES INTO THE TABLE
SQL> insert into stud values ('ashwin',101,19,'anna

nagar','aeronautical'); 1 row created.

SQL> insert into stud values ('bhavesh',102,18,'nungambakkam','marine');

1 row created.

SQL> insert into stud values ('pruthvik',103,20,'anna

nagar','aerospace'); 1 row created.

SQL> insert into stud values
('charith',104,20,'kilpauk','mechanical'); 1 row created.

SQL> select * from stud;
SNAME SID SAGE SAREA SDEPT ------------------------------ ----------
--------- -------------------- -------------------- ashwin 101 19 anna nagar
aeronautical bhavesh 102 18 nungambakkam marine pruthvik 103 20
anna nagar aerospace charith 104 20 kilpauk mechanical **RENAMING
THE TABLE 'STUD'**
SQL> rename stud to studs;

Table renamed.

ARITHMETIC OPERATION

SQL> select sname, sid+100 "stid" from studs;

```
SNAME stid
---------------------------- ---------
ashwin 201
bhavesh 202
pruthvik 203
charith 204
```

CONCATENATION OPERATOR

```
SQL> select sname || ' is a ' || sdept || ' engineer. ' AS "PROFESSION" from studs;

PROFESSION
-------------------------------------------------------------------
ashwin is a aeronautical engineer.
bhavesh is a marine engineer.
pruthvik is a aerospace engineer.
charith is a mechanical engineer.
```

DISPLAY ONLY DISTINCT VALUES

```
SQL> select distinct sarea from studs;

SAREA
--------------------
anna nagar
kilpauk
nungambakkam
```

USING THE WHERE CLAUSE

```
SQL> select sname,sage from studs where sage<=19;

SNAME SAGE
---------------------------- ---------
ashwin 19
bhavesh 18
```

BETWEEN OPERATOR

```
SQL> select sname,sarea, sid from studs where sid between 102 and 104;

SNAME SAREA SID
---------------------------- -------------------- ----------
bhavesh nungambakkam 102
pruthvik anna nagar 103
charith kilpauk 104
```

IN PREDICATE

```
SQL> select sname,sarea , sid from studs where sid in(102,104);
```

```
SNAME SAREA SID
------------------------------ -------------------- ----------
bhavesh nungambakkam 102
charith kilpauk 104
```
PATTERN MATCHING

```
SQL> select sname, sarea from studs where sarea like '%g%';

SNAME SAREA
------------------------------ --------------------
ashwin anna nagar
bhavesh nungambakkam
pruthvik anna nagar
```

LOGICAL AND OPERATOR

```
SQL> select sname ,sid from studs where sid>102 and sarea='anna nagar';

SNAME SID
------------------------------ ----------
pruthvik 103
```

LOGICAL OR OPERATOR

```
SQL> select sname ,sid from studs where sid>102 or sarea='anna nagar';

SNAME SID
------------------------------ ----------
ashwin 101
pruthvik 103
charith 104
```
NOT IN PREDICATE

```
SQL> select sname, sid from studs where sid not in(102,104);

SNAME SID
------------------------------ ----------
ashwin 101
pruthvik 103
```

UPDATING THE TABLE

```
SQL> alter table studs add ( spocket varchar2(20) );

Table altered.

SQL> update studs set spocket=750 where sid=101;

1 row updated.
```

```
SQL> update studs set spocket=500 where sid=102;

1 row updated.

SQL> update studs set spocket=250 where sid=103;
1 row updated.

SQL> update studs set spocket=100 where sid=104;

1 row updated.

SQL> select * from studs;


SNAME SID SAGE SAREA SDEPT ------------------------------ ----------
--------- -------------------- -------------------- SPOCKET
--------------------

ashwin 101 19 anna nagar aeronautical  750
bhavesh 102 18 nungambakkam marine 500
pruthvik 103 20 anna nagar aerospace 250
charith 104 20 kilpauk mechanical 100
```

AGGREGATE FUNCTIONS

```
SQL> select avg( spocket ) result from studs;

 RESULT
---------
 400

SQL> select min(spocket) result from studs;

RESULT
--------------------
100

SQL> select count(spocket) result from studs;

 RESULT
---------
 4

SQL> select count(*) result from studs;

 RESULT
```

```
---------
 4

SQL> select count(spocket) result from studs where sarea='anna nagar';

 RESULT
---------
 2
SQL> select max(spocket) result from studs;

RESULT
--------------------
750

SQL> select sum(spocket) result from studs;

 RESULT
---------
 1600
```

NUMERIC FUNCTIONS

```
SQL> select abs(-20) result from dual;

 RESULT
---------
 20

SQL> select power (2,10) result from dual;

 RESULT
---------
 1024

SQL> select round(15.359,2) result from dual;

 RESULT
---------
  15.36

SQL> select sqrt (36) result from dual;

 RESULT
---------
 6
```

STRING FUNCTIONS

```
SQL> select lower('ORACLE') result from dual;
```

```
RESULT
------
oracle

SQL> select upper('oracle') result from dual;

RESULT
------
ORACLE

SQL> select initcap('Oracle') result from dual;

RESULT
------
Oracle
SQL> select substr('oracle' ,2 ,5) result from dual;

RESULT
-----
racle

SQL> select lpad('oracle',10,'#') result from dual;

RESULT
----------
####oracle

SQL> select rpad ('oracle',10,'^') result from dual;

RESULT
----------
oracle^^^^
```

CONVERSION FUNCTIONS

```
SQL> update studs set sage=to_number(substr(118,2,3));

4 rows updated.

SQL> select * from studs;

SNAME SID SAGE SAREA SDEPT ------------------------------ ----------
--------- -------------------- --------------------
SPOCKET
--------------------
```

```
ashwin 101 18 anna nagar aeronautical 750
bhavesh 102 18 nungambakkam marine 500
pruthvik 103 18 anna nagar aerospace 250
charith 104 18 kilpauk mechanical 100

SQL> select to_char( 17145, '099,999') result from dual;

RESULT
--------
 017,145

SQL> select to_char(sysdate,'dd-mon-yyyy') result from dual;

RESULT
-----------
16-jul-2008
```

DATE FUNCTIONS

```
SQL> select sysdate from dual;

SYSDATE
---------
16-JUL-08

SQL> select sysdate,add_months(sysdate,4) result from dual;

SYSDATE RESULT
--------- ---------
16-JUL-08 16-NOV-08

SQL> select sysdate, last_day(sysdate) result from dual;

SYSDATE RESULT
--------- ---------
16-JUL-08 31-JUL-08

SQL> select sysdate, next_day(sysdate,'sunday') result from dual;

SYSDATE RESULT
--------- ---------
16-JUL-08 20-JUL-08

SQL> select months_between('09-aug-91','11-mar-90') result from dual;

 RESULT
 ---------
 16.935484
```

GROUP BY CLAUSE

```
SQL> select sarea, sum(spocket) result from studs group by sarea;

SAREA RESULT
-------------------- ------------
anna nagar 1000
nungambakkam 500
kilpauk 100
```

HAVING CLAUSE

```
SQL> select sarea, sum(spocket) result from studs group by sarea having spocket<600;

SAREA RESULT
-------------------- ------------
nungambakkam 500
kilpauk 100
```

DELETION

```
SQL> delete from studs where sid=101;

1 row deleted.

SQL> select * from studs;

SNAME SID SAGE SAREA SDEPT
------------------------------ ---------- --------- --------------------
-------------------- SPOCKET
-------------------
bhavesh 102 18 nungambakkam marine 500
pruthvik 103 20 anna nagar aerospace 250
charith 104 20 kilpauk mechanical
100
```

CREATING TABLES FOR DOING SET OPERATIONS

TO CREATE PRODUCT TABLE

```
SQL> create table product(prodname varchar2(30), prodno varchar2(10));
Table created.

SQL> insert into product values('table',10001);

1 row created.

SQL> insert into product values('chair',10010);
```

```
1 row created.

SQL> insert into product values('desk',10110);

1 row created.

SQL> insert into product values('cot',11110);

1 row created.

SQL> insert into product values('sofa',10010);

1 row created.

SQL>

SQL> insert into product values('tvstand',11010);

1 row created.

SQL> select * from product;

PRODNAME PRODNO
------------------------------ ----------
table 10001
chair 10010
desk 10110
cot 11110
sofa 10010
tvstand 11010
```

TO CREATE SALE TABLE

```
SQL> create table sale(prodname varchar2(30),orderno number(10),prodno
varchar2(10));

Table created.
SQL> insert into sale values('table',801,10001);

1 row created.

SQL> insert into sale values('chair',805,10010);

1 row created.

SQL> insert into sale values('desk',809,10110);
```

```
1 row created.

SQL> insert into sale values('cot',813,11110);

1 row created.

SQL> insert into sale values('sofa',817,10010);

1 row created.

SQL> select * from sale;

PRODNAME ORDERNO PRODNO
------------------------------ --------- ----------
table 801 10001
chair 805 10010
desk 809 10110
cot 813 11110
sofa 817 10010
```

SET OPERATIONS

```
SQL> select prodname from product where prodno=10010 union select prodname from
sale where prodno=10010;

PRODNAME
------------------------------
chair
sofa

SQL> select prodname from product where prodno=11110 intersect select prodname
from sale where prodno=11110;

PRODNAME
------------------------------
cot
```

CREATING TABLES FOR DOING JOIN AND NESTED QUERY OPERATIONS

TO CREATE SSTUD1 TABLE

```
SQL> create table sstud1 ( sname varchar2(20) , place varchar2(20));

Table created.

SQL> insert into sstud1 values ( 'prajan','chennai');

1 row created.
```

```
SQL> insert into sstud1 values ( 'anand','chennai');

1 row created.

SQL> insert into sstud1 values ( 'kumar','chennai');

1 row created.

SQL> insert into sstud1 values ( 'ravi','chennai');

1 row created.

SQL> select * from sstud1;

SNAME PLACE
-------------------- --------------------
prajan chennai
anand chennai
kumar chennai
ravi chennai
```

TO CREATE SSTUD2 TABLE

```
SQL> create table sstud2 ( sname varchar2(20), dept varchar2(10), marks number(10));

Table created.

SQL> insert into sstud2 values ('prajan','cse',700);

1 row created.

SQL> insert into sstud2 values ('anand','it',650);

1 row created.

SQL> insert into sstud2 values ('vasu','cse',680);

1 row created.
SQL> insert into sstud2 values ('ravi','it',600);

1 row created.

SQL> select * from sstud2;

SNAME DEPT MARKS
-------------------- ---------- ---------
```

```
prajan cse 700
anand it 650
vasu cse 680
ravi it 600
```

JOIN OPERATIONS

```
SQL> select sstud1.sname, dept from sstud1 inner join sstud2 on ( sstud1.sname=
sstud2.sname);

SNAME DEPT
-------------------- ----------
anand it
prajan cse
ravi it

SQL> select sstud1.sname, dept from sstud1 join sstud2 on ( sstud1.sname=
sstud2.sname);

SNAME DEPT
-------------------- ----------
anand it
prajan cse
ravi it

SQL> select sstud1.sname, dept from sstud1 left outer join sstud2 on ( sstud1.sname=
sstud2.sname);

SNAME DEPT
-------------------- ----------
prajan cse
anand it
ravi it
kumar

SQL> select sstud1.sname, dept from sstud1 right outer join sstud2 on ( sstud1.sname=
sstud2.sname)
SNAME DEPT
-------------------- ----------
prajan cse
anand it
ravi it
 cse

SQL> select sstud1.sname, dept from sstud1 full outer join sstud2 on ( sstud1.sname=
sstud2.sname);
```

```
SNAME DEPT
-------------------- ----------
prajan cse
anand it
ravi it
kumar
 cse
```

NESTED QUERIES

```
SQL> select sname from sstud1 where sstud1.sname in ( select sstud2.sname from
2 sstud2 );

SNAME
--------------------
anand
prajan
ravi

SQL> select sname from sstud1 where sstud1.sname not in ( select sstud2.sname from
sstud2 );

SNAME
--------------------
kumar

SQL> select sname from sstud2 where marks > some(select marks from sstud2
2 where dept='cse');

SNAME
--------------------
prajan
SQL> select sname from sstud2 where marks >= some (select marks from sstud2
2 where dept='cse' );

SNAME
--------------------
prajan
vasu

SQL> select sname from sstud2 where marks > any ( select marks from sstud2 where
dept='cse' );

SNAME
--------------------
```

```
prajan

SQL> select sname from sstud2 where marks >= any ( select marks from sstud2
2 where dept='cse' );

SNAME
--------------------
prajan
vasu

SQL> select sname from sstud2 where marks > all ( select marks from sstud2 where
dept='cse' );

no rows selected

SQL> select sname from sstud2 where marks < all ( select marks from sstud2 where
dept='cse' );

SNAME
--------------------
anand
ravi

SQL> select sname from sstud1 where exists ( select sstud2.sname from sstud2
2 where sstud1.sname=sstud2.sname );

SNAME
--------------------
prajan
anand
ravi

SQL> select sname from sstud1 where not exists ( select sstud2.sname from
 2 sstud2 where sstud1.sname=sstud2.sname );

SNAME
--------------------
kumar
```

RESULT

Thus all the DML commands were executed and the output was verified.

INTEGRITY CONSTRAINTS

EX NO: 03

DATE:

AIM

To study the various constraints available in the SQL query language.

DOMAIN INTEGRITY CONSTRAINTS

NOT NULL CONSTRAINT

```
SQL> create table empl (ename varchar2(30) not null, eid varchar2(20) not null);

Table created.

SQL> insert into empl values ('abcde',11);

1 row created.

SQL> insert into empl values ('fghij',12);

1 row created.

SQL> insert into empl values ('klmno',null);
insert into empl values ('klmno',null)
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("ITA"."EMPL"."EID")


SQL> select * from empl;

ENAME EID
----------------------------- --------------------
abcde 11
fghij 12
```

CHECK AS A COLUMN CONSTRAINT

```
SQL> create table depts ( dname varchar2(30) not null, did number(20) not null check
(did<10000));

Table created.

SQL> insert into depts values ('sales ',9876);

1 row created.
SQL> insert into depts values ('marketing',5432);
```

```
1 row created.

SQL> insert into depts values ('accounts',789645);
insert into depts values ('accounts',789645)
*
ERROR at line 1:
ORA-02290: check constraint (ITA.SYS_C003179) violated


SQL> select * from depts;

DNAME DID
------------------------------ ---------
sales 9876
marketing 5432
```

CHECK AS A TABLE CONSTRAINT

```
SQL> create table airports (aname varchar2(30) not null , aid number(20) not null, acity
varchar2(30) check( acity in ('chennai','hyderabad','bangalore')));

Table created.

SQL> insert into airports values( 'abcde', 100,'chennai');

1 row created.

SQL> insert into airports values( 'fghij', 101,'hyderabad');

1 row created.

SQL> insert into airports values( 'klmno', 102,'bangalore');

1 row created.

SQL> insert into airports values( 'pqrst', 103,'mumbai');
insert into airports values( 'pqrst', 103,'mumbai')
*
ERROR at line 1:
ORA-02290: check constraint (ITA.SYS_C003187) violated


SQL> select * from airports;
ANAME AID ACITY
------------------------------ --------- ------------------------------
abcde 100 chennai
fghij 101 hyderabad
```

```
klmno 102 bangalore
```

<u>ENTITY INTEGRITY CONSTRAINTS</u>

UNIQUE AS A COLUMN CONSTRAINT

```
SQL> create table book (bname varchar2(30) not null, bid number(20) not null unique);

Table created.

SQL> insert into book values ('fairy tales',1000);

1 row created.

SQL> insert into book values ('bedtime stories',1001);

1 row created.

SQL> insert into book values ('comics',1001);
insert into book values ('comics',1001)
*
ERROR at line 1:
ORA-00001: unique constraint (ITA.SYS_C003130) violated


SQL> select * from book;

BNAME BID
----------------------------- ---------
fairy tales 1000
bedtime stories 1001
```

UNIQUE AS A TABLE CONSTRAINT

```
SQL> create table orders( oname varchar2(30) not null , oid number(20) not null ,
unique(oname,oid));

Table created.

SQL> insert into orders values ('chair', 2005);
1 row created.

SQL> insert into orders values ('table',2006);

1 row created.
```

```
SQL> insert into orders values ('chair',2007);

1 row created.

SQL> insert into orders values ('chair', 2005);
insert into orders values ('chair', 2005)
*
ERROR at line 1:
ORA-00001: unique constraint (ITA.SYS_C003152) violated

SQL> select * from orders;

ONAME OID
------------------------------ ---------
chair 2005
table 2006
chair 2007
```

PRIMARY KEY AS A COLUMN CONSTRAINT

```
SQL> create table custo ( cname varchar2(30) not null , cid number(20) not null
primary key);

Table created.

SQL> insert into custo values ( 'jones', 506);

1 row created.

SQL> insert into custo values ('hayden',508);

1 row created.

SQL> insert into custo values ('ricky',506);
insert into custo values ('ricky',506)
*
ERROR at line 1:
ORA-00001: unique constraint (ITA.SYS_C003165) violated
SQL> select * from custo;

CNAME CID
------------------------------ ---------
jones 506
hayden 508
```

PRIMARY KEY AS A TABLE CONSTRAINT

```
SQL> create table branches( bname varchar2(30) not null , bid number(20) not null ,
primary key(bname,bid));

Table created.

SQL> insert into branches values ('anna nagar', 1005);

1 row created.

SQL> insert into branches values ('adyar',1006);

1 row created.

SQL> insert into branches values ('anna nagar',1007);

1 row created.

SQL> insert into branches values ('anna nagar', 1005);
insert into branches values ('anna nagar', 1005)
*
ERROR at line 1:
ORA-00001: unique constraint (ITA.SYS_C003173) violated


SQL> select * from branches;


BNAME BID
----------------------------- ---------
anna nagar 1005
adyar 1006
anna nagar 1007
```

REFERENTIAL INTEGRITY CONSTRAINTS

TO CREATE 'DEPTS' TABLE

```
SQL> create table depts(city varchar2(20), dno number(5) primary
key); Table created.
SQL> insert into depts values('chennai', 11);
1 row created.
SQL> insert into depts values('hyderabad', 22);
1 row created.
```

TO CREATE 'SEMP' TABLE

```
SQL> create table semp(ename varchar2(20), dno number(5) references depts(dno));

Table created.
SQL> insert into semp values('x', 11);
1 row created.
SQL> insert into semp values('y', 22);
1 row created.
SQL> insert into semp values('z', 33);
insert into semp values('z', 33)
*
ERROR at line 1:
ORA-00001: referential integrity constraint (ITA.SYS_C003273) violated


SQL> select * from semp;
ENAME DNO
-------------------- ---------
x 11
y 22
```

ALTER TABLE

```
SQL> alter table semp add(eddress varchar2(20));
Table altered.
SQL> update semp set eddress='10 gandhi road' where
dno=11; 1 row updated.
SQL> update semp set eddress='12 m.g. road' where dno=22;
1 row updated.

SQL> select * from semp;
ENAME DNO EDDRESS
-------------------- --------- --------------------
x 11 10 gandhi road
y 22 12 m.g. road

SQL> select city, ename from depts, s2emp where depts.dno =
s2emp.dno; CITY ENAME
-------------------- --------------------
chennai x
hyderabad y
```

RESULT

Thus the various constraints were implemented and the tables were created using the respecting constraints. Hence the output was verified.

VIEWS

EX NO: 4
DATE:

AIM

To create views for the table and perform operations on it.

DEFINITION

A view is an object that gives the user the logical view of data from the underlying table.
Any relation that is not part of the logical model but is made visible to the user as a virtual relation is called a view. They are generally used to avoid duplication of data.

Views are created for the following reasons,

- Data simplicity
- To provide data security
- Structural simplicity (because view contains only limited number of rows and columns)

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 STRUTURE OF THE TABLE AND THE VIEW ARE NOT SIMILAR -- PROOF

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

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 STRUTURE OF THE TABLE AND THE VIEW 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 SAL DNO
```

```
-------------------- --------- --------- ---------
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')
*

ERROR at line 1:
ORA-01779: cannot modify a column which maps to a non key-preserved table
```

This shows that when a view is created from two different tables no manipulations can be performed using that view and the above error is displayed.

RESULT

Thus views were created, various operations were performed and the outputs were verified.

DATACONTROL LANGUAGE COMMANDS

EX NO: 5
DATE:

AIM

To study the various data language commands (DCL) and implement them on the database.

DESCRIPTION

The DCL language is used for controlling the access to the table and hence securing the database. This language is used to provide certain priveleges to a particular user. Priveleges are rights to be allocated. The privilege commands are namely,

- Grant
- Revoke

The various privileges that can be granted or revoked are,

- Select
- Insert
- Delete
- Update
- References
- Execute
- All

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 their database objects to other users.

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

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 for tables.
<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.

EXAMPLES

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.
```

RESULT

Thus all the commands were executed and their outputs were verified.

TRANSACTION CONTROL LANGUAGE

EX NO: 6
DATE:

AIM

To study the various TCL commands namely commit, rollback and savepoint.

DESCRIPTION

COMMIT: This command saves all the transactions to the database since the last commit or rollback command.
ROLLBACK: This command is used to undo the transactions that have not been already saved to the database.It can be used to undo transactions since the last commit or rollback command.
SAVEPOINT: This command is a point in transaction that you can roll the transaction back to without rolling back the entire transmission.

CREATE THE TABLE 'ITYR'

```
SQL> create table ityr(ename varchar(15),eid number(5),salary

number(5)); Table created.
```

PROGRAM

```
SQL> set serveroutput on;
SQL> declare
 2 t number(6);
 3 n number(6);
 4 s number(6);
 5 begin
 6 insert into ityr values('a',100,19000);
 7 insert into ityr values('b',102,1000);
 8 s:=&s;
 9 n:=&n;
 10 savepoint a;
 11 update ityr set salary=salary+2000 where eid=s;
 12 update ityr set salary=salary+1500 where eid=n;
 13 select sum(salary) into t from ityr;
 14 if(t>20000)
 15 then
 16 rollback to a;
 17 else
 18 dbms_output.put_line('no updation');
 19 end if;
 20 end ;
 21 /
Enter value for s: 100
old 8: s:=&s;
new 8: s:=100;
Enter value for n: 102
old 9: n:=&n;
new 9: n:=102;

PL/SQL procedure successfully completed.
```

DISPLAYING THE UPDATED TABLE

```
SQL> select * from ityr;

ENAME EID SALARY
--------------- ---------- ----------
a 100 19000
b 102 1000
```

RESULT

Thus the various commands were executed and the output was verified.

PROCEDURAL LANGUAGE/ STRUCTURAL QUERY LANGUAGE

EX NO: 7

DATE:

AIM

To implement various programs using PL/SQL language.

PROGRAMS

TO DISPLAY HELLO MESSAGE

```
SQL> set serveroutput on;
SQL> declare
 2 a varchar2(20);
 3 begin
 4 a:='Hello';
 5 dbms_output.put_line(a);
 6 end;
 7 /
Hello

PL/SQL procedure successfully completed.
```

TO INPUT A VALUE FROM THE USER AND DISPLAY IT

```
SQL> set serveroutput on;
SQL> declare
 2 a varchar2(20);
 3 begin
 4 a:=&a;
 5 dbms_output.put_line(a);
 6 end;
 7 /
Enter value for a: 5
old 4: a:=&a;
new 4: a:=5;
5

PL/SQL procedure successfully completed.
```

GREATEST OF TWO NUMBERS

```
SQL> set serveroutput on;
SQL> declare
 2 a number(7);
 3 b number(7);
 4 begin
 5 a:=&a;
 6 b:=&b;
 7 if(a>b) then
 8 dbms_output.put_line (' The grerater of the two is'|| a);
9 else
 10 dbms_output.put_line (' The grerater of the two is'|| b);
 11 end if;
 12 end;
 13 /
Enter value for a: 5
old 5: a:=&a;
new 5: a:=5;
Enter value for b: 9
old 6: b:=&b;
new 6: b:=9;
The grerater of the two is9

PL/SQL procedure successfully completed.
```

GREATEST OF THREE NUMBERS

```
SQL> set serveroutput on;
SQL> declare
 2 a number(7);
 3 b number(7);
 4 c number(7);
 5 begin
 6 a:=&a;
 7 b:=&b;
 8 c:=&c;
 9 if(a>b and a>c) then
 10 dbms_output.put_line (' The greatest of the three is ' || a);
 11 else if (b>c) then
 12 dbms_output.put_line (' The greatest of the three is ' || b);
 13 else
 14 dbms_output.put_line (' The greatest of the three is ' || c);
 15 end if;
 16 end if;
 17 end;
 18 /
Enter value for a: 5
old 6: a:=&a;
new 6: a:=5;
Enter value for b: 7
old 7: b:=&b;
new 7: b:=7;
Enter value for c: 1
old 8: c:=&c;
new 8: c:=1;
```

```
The greatest of the three is 7

PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 5 USING SIMPLE LOOP

```
SQL> set serveroutput on;
SQL> declare
 2 a number:=1;
 3 begin
 4 loop
 5 dbms_output.put_line (a);
 6 a:=a+1;
 7 exit when a>5;
 8 end loop;
 9 end;
 10 /
1
2
3
4
5

PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 4 USING WHILE LOOP

```
SQL> set serveroutput on;
SQL> declare
 2 a number:=1;
 3 begin
 4 while(a<5)
 5 loop
 6 dbms_output.put_line (a);
 7 a:=a+1;
 8 end loop;
 9 end;
 10 /
1
2
3
4
PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 5 USING FOR LOOP

```
SQL> set serveroutput on;
SQL> declare
 2 a number:=1;
 3 begin
 4 for a in 1..5
 5 loop
 6 dbms_output.put_line (a);
 7 end loop;
 8 end;
 9 /
1
2
3
4
5

PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 5 IN REVERSE ORDER USING FOR
LOOP SQL> set serveroutput on;

```
SQL> declare
 2 a number:=1;
 3 begin
 4 for a in reverse 1..5
 5 loop
 6 dbms_output.put_line (a);
 7 end loop;
 8 end;
 9 /
5
4
3
2
1

PL/SQL procedure successfully completed.
```

TO CALCULATE AREA OF CIRCLE

```
SQL> set serveroutput on;
SQL> declare
 2 pi constant number(4,2):=3.14;
 3 a number(20);
 4 r number(20);
 5 begin
 6 r:=&r;
 7 a:= pi* power(r,2);
 8 dbms_output.put_line (' The area of circle is ' || a);
 9 end;
 10 /
```

```
Enter value for r: 2
old 6: r:=&r;
new 6: r:=2;
The area of circle is 13

PL/SQL procedure successfully completed.
```

TO CREATE SACCOUNT TABLE

```
SQL> create table saccount ( accno number(5), name varchar2(20), bal number(10));

Table created.


SQL> insert into saccount values ( 1,'mala',20000);

1 row created.

SQL> insert into saccount values (2,'kala',30000);

1 row created.

SQL> select * from saccount;

 ACCNO NAME BAL
--------- ------------------- ---------
 1 mala 20000
 2 kala 30000

SQL> set serveroutput on;
SQL> declare
 2 a_bal number(7);
 3 a_no varchar2(20);
 4 debit number(7):=2000;
 5 minamt number(7):=500;
 6 begin
 7 a_no:=&a_no;
 8 select bal into a_bal from saccount where accno= a_no;
 9 a_bal:= a_bal-debit;
 10 if (a_bal > minamt) then
 11 update saccount set bal=bal-debit where accno=a_no;
 12 end if;
 13 end;
 14
 15 /
Enter value for a_no: 1
old 7: a_no:=&a_no;
new 7: a_no:=1;
```

```
PL/SQL procedure successfully completed.




SQL> select * from saccount;

 ACCNO NAME BAL
--------- ------------------- ---------
 1 mala 18000
 2 kala 30000
```

TO CREATE TABLE SROUTES

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination
varchar2(20), fare numbe
r(10), distance number(10));

Table created.

SQL> insert into sroutes values ( 2, 'chennai', 'dindugal',

400,230); 1 row created.

SQL> insert into sroutes values ( 3, 'chennai', 'madurai',

250,300); 1 row created.

SQL> insert into sroutes values ( 6, 'thanjavur', 'palani',

350,370); 1 row created.

SQL> select * from sroutes;

 RNO ORIGIN DESTINATION FARE DISTANCE ---------
-------------------- -------------------- --------- ---------
 2 chennai dindugal 400 230
 3 chennai madurai 250 300
 6 thanjavur palani 350 370


SQL> set serveroutput on;
SQL> declare
 2 route sroutes.rno % type;
 3 fares sroutes.fare % type;
 4 dist sroutes.distance % type;
 5 begin
 6 route:=&route;
 7 select fare, distance into fares , dist from sroutes where rno=route;
8 if (dist < 250) then
 9 update sroutes set fare=300 where rno=route;
 10 else if dist between 250 and 370 then
 11 update sroutes set fare=400 where rno=route;
 12 else if (dist > 400) then
 13 dbms_output.put_line('Sorry');
 14 end if;
 15 end if;
 16 end if;
 17 end;
 18 /
Enter value for route: 3
old 6: route:=&route;
new 6: route:=3;

PL/SQL procedure successfully completed.

SQL> select * from sroutes;

 RNO ORIGIN DESTINATION FARE DISTANCE ---------
-------------------- -------------------- --------- ---------
 2 chennai dindugal 400 230
 3 chennai madurai 400 300
 6 thanjavur palani 350 370
```

TO CREATE SCA LCULATE TABLE

```
SQL> create table scalculate ( radius number(3), area

number(5,2)); Table created.

SQL> desc scalculate;
 Name Null? Type
 ----------------------------------------------------- --------
 ------------------------------------ RADIUS NUMBER(3)
 AREA NUMBER(5,2)
SQL> set serveroutput on;
SQL> declare
 2 pi constant number(4,2):=3.14;
 3 area number(5,2);
 4 radius number(3);
 5 begin
 6 radius:=3;
 7 while (radius <=7)
 8 loop
 9 area:= pi* power(radius,2);
 10 insert into scalculate values (radius,area);
 11 radius:=radius+1;
 12 end loop;
 13 end;
 14 /

PL/SQL procedure successfully

completed. SQL> select * from scalculate;

 RADIUS AREA
--------- ---------
 3 28.26
 4 50.24
 5 78.5
 6 113.04
 7 153.86
```

TO CALCULATE FACTORIAL OF A GIVEN NUMBER

```
SQL> set serveroutput on;
SQL> declare
 2 f number(4):=1;
 3 i number(4);
 4 begin
 5 i:=&i;
 6 while(i>=1)
 7 loop
 8 f:=f*i;
 9 i:=i-1;
 10 end loop;
 11 dbms_output.put_line('The value is ' || f);
 12 end;
 13 /
Enter value for i: 5
old 5: i:=&i;
new 5: i:=5;
The value is 120

PL/SQL procedure successfully completed.
```

RESULT

Thus the various programs were implemented and their output was verified.

TRIGGERS

EX NO: 8
DATE:

AIM

To study and implement the concept of triggers.

DEFINITION

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are, • **Trigger statement**: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.

• **Trigger body or trigger action**: It is a PL/SQL block that is executed when the triggering statement is used.

• **Trigger restriction**: Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

• To generate data automatically
• To enforce complex integrity constraints
• To customize complex securing authorizations
• To maintain the replicate table
• To audit data modifications

TYPES OF TRIGGERS

The various types of triggers are as follows,

• **Before**: It fires the trigger before executing the trigger statement. •
After: It fires the trigger after executing the trigger statement.
• **For each row**: It specifies that the trigger fires once per row.
• **For each statement**: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

VARIABLES USED IN TRIGGERS

• :new
• :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

SYNTAX

```
 create or replace trigger triggername [before/after] {DML
 statements} on [tablename] [for each row/statement]
 begin
 -------------------------
 -------------------------
 -------------------------
exception
end;
```

USER DEFINED ERROR MESSAGE

The package “raise_application_error” is used to issue the user defined error messages
Syntax: raise_application_error(error number,‘error message‘);
The error number can lie between -20000 and -20999.
The error message should be a character string.

TO CREATE THE TABLE ‘ITEMPLS’

```
SQL> create table itempls (ename varchar2(10), eid number(5), salary number(10));

Table created.

SQL> insert into itempls values('xxx',11,10000);

1 row created.

SQL> insert into itempls values('yyy',12,10500);

1 row created.

SQL> insert into itempls values('zzz',13,15500);

1 row created.

SQL> select * from itempls;
```

```
ENAME EID SALARY
---------- --------- ---------
xxx 11 10000
yyy 12 10500
zzz 13 15500
```

TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each
row  2 begin
 3 raise_application_error(-20010,'You cannot do manipulation');
4 end;
 5
 6 /

Trigger created.

SQL> insert into itempls values('aaa',14,34000);
insert into itempls values('aaa',14,34000)
 *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'
 *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

SQL> update itempls set eid=15 where ename='yyy';
update itempls set eid=15 where ename='yyy'
 *
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger
```

'STUDENT.ITTRIGG' **TO DROP THE CREATED TRIGGER**

```
SQL> drop trigger ittrigg;
```

```
Trigger dropped.
```

TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE AND DOES NOT ALLOW UPDATION AND INSERTION

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
 2 declare
 3 triggsal itempls.salary%type;
 4 begin
 5 select salary into triggsal from itempls where eid=12;
 6 if(:new.salary>triggsal or :new.salary<triggsal) then
 7 raise_application_error(-20100,'Salary has not been changed');
8 end if;
 9 end;
 10 /

Trigger created.


SQL> insert into itempls values ('bbb',16,45000);
insert into itempls values ('bbb',16,45000)
 *
ERROR at line 1:
 ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation


SQL> update itempls set eid=18 where ename='zzz';
update itempls set eid=18 where ename='zzz'
 *
ERROR at line 1:
 ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
```

RESULT

Thus the triggers were created , executed and their respective outputs were verified.

GOTO AND EXCEPTIONS

EX NO: 9
DATE:

AIM

To perform goto and exception handling mechanisms.

GOTO COMMAND

PURPOSE

The GOTO statement changes the flow of control within a PL/SQL block. The entry point into such a block of code is marked using the tags. This statement makes use of the
<<user defined name>> to jump into the block of code for execution.

SYNTAX

GOTO <code block name> <<user defined name>>

CREATING THE TABLES 'SPRODUCTMASTERS' AND 'SOLDPRICES'

```
SQL> create table sproductmasters( pno varchar2(10), sellprice number(10));

Table created.

SQL> insert into sproductmasters values('p1',3200);

1 row created.

SQL> insert into sproductmasters values('p2',4000);

1 row created.

SQL> insert into sproductmasters values('p3',6000);

1 row created.

SQL> select * from sproductmasters;

PNO SELLPRICE
---------- ---------
p1 3200
p2 4000
p3 6000

SQL> create table soldprices( pno varchar2(10), datechange varchar2(20),soldprices
number(10));

Table created.
```

OPERATION TO BE PERFORMED

If the price of a product is less than 4000 then change to 4000. The price change is to be recorded on the old price table along with the product number and the date on which the price was last changed using PL/SQL.

PROGRAM

```
declare
sellingprice number(10,2);
begin
select sellprice into sellingprice from sproductmasters where pno='p1';
if sellingprice < 4000
then
goto add_old_price;
else
dbms_output.put_line(' Current price is '|| sellingprice);
end if;
<<add_old_price>>
update sproductmasters set sellprice = 4000 where pno='p1';
insert into soldprices values('p1',sysdate,sellingprice);
dbms_output.put_line(' The new price of p1 is 4000 ');
end;
/
```

PROGRAM OUTPUT

```
The new price of p1 is 4000

PL/SQL procedure successfully completed.
```

DISPLAYING THE CONTENTS OF 'SOLDPRICES' TABLE

```
SQL> select * from soldprices;

PNO DATECHANGE SOLDPRICES
---------- -------------------- ----------
p1 27-AUG-08 3200
```

EXCEPTIONS

Exceptions are error handling mechanisms. They are of 2 types,

- Pre – defined exceptions
- User – defined exceptions

TO CREATE THE TABLE 'SSITEMS' ON WHICH THE EXCEPTION HANDLING MECHANISMS ARE GOING TO BE PERFORMED

```
SQL> create table ssitems( id number(10), quantity number(10), actualprice
number(10));

Table created.

SQL> insert into ssitems values(100,5,5000);
```

```
1 row created.

SQL> insert into ssitems values(101,6,9000);

1 row created.

SQL> insert into ssitems values(102,4,4000);

1 row created.

SQL> insert into ssitems values(103,2,2000);

1 row created.

SQL> select * from ssitems;

 ID QUANTITY ACTUALPRICE
--------- --------- -----------
 100 5 5000
 101 6 9000
 102 4 4000
 103 2 2000
```

PRE – DEFINED EXCEPTIONS

SYNTAX

```
 begin
 sequence of statements;
 exception
 when < exception name > then
 sequence of statements;
 end;
```

EXAMPLE USING PL/SQL

```
SQL> set serveroutput on;
SQL> declare
 2 price ssitems.actualprice % type;
 3 begin
 4 select actualprice into price from ssitems where quantity=10;
5 exception
 6 when no_data_found then
 7 dbms_output.put_line ('ssitems missing');
 8 end;
 9 /
ssitems missing

PL/SQL procedure successfully completed.
```

DISPLAYING THE UPDATED TABLE

```
SQL> select * from ssitems;

 ID QUANTITY ACTUALPRICE
--------- --------- -----------
 100 5 5000
 101 6 9000
 102 4 4000
 103 2 2000
```

USER DEFINED EXCEPTONS

SYNTAX

```
 declare
 < exception name > exception;
begin
 sequence of statements;
 raise < exception name >;
 exception
 when < exception name > then
 sequence of statements;
 end;
```

EXAMPLE USING PL/SQL

```
SQL> set serveroutput on;
SQL> declare
 2 zero_price exception;
 3 price number(8,2);
 4 begin
 5 select actualprice into price from ssitems where id=103;
6 if price=0 or price is null then
 7 raise zero_price;
 8 end if;
 9 exception
 10 when zero_price then
 11 dbms_output.put_line('Failed zero price');
 12 end;
 13 /
```

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

DISPLAYING THE UPDATED TABLE

```
SQL> select * from ssitems;

 ID QUANTITY ACTUALPRICE
--------- --------- -----------
 100 5 5000
 101 6 9000
 102 4 4000
 103 2 2000
```

RESULT

Thus the goto statement and exceptions were executed and their respective outputs were verified.

CURSORS

EX NO: 10
DATE:

AIM

To write PL/SQL blocks that implement the concept of for the 3 types of cursors namely,

- Cursor for loop
- Explicit cursor
- Implicit cursor

TO CREATE THE TABLE 'SSEMPP'

```
SQL> create table ssempp( eid number(10), ename varchar2(20), job varchar2(20), sal
number (10),dnonumber(5));

Table created.

SQL> insert into ssempp values(1,'nala','lecturer',34000,11);

1 row created.

SQL> insert into ssempp values(2,'kala',' seniorlecturer',20000,12);
```

```
1 row created.

SQL> insert into ssempp values(5,'ajay','lecturer',30000,11);

1 row created.

SQL> insert into ssempp values(6,'vijay','lecturer',18000,11);

1 row created.

SQL> insert into ssempp values(3,'nila','professor',60000,12);

1 row created.

SQL> select * from ssempp;

 EID ENAME JOB SAL DNO
--------- ------------------- ------------------- --------- ---------
 1 nala lecturer 34000 11
 2 kala seniorlecturer 20000 12
 5 ajay lecturer 30000 11
 6 vijay lecturer 18000 11
 3 nila professor 60000 12
```

TO WRITE A PL/SQL BLOCK TO DISPLAY THE EMPOYEE ID AND EMPLOYEE NAME USING CURSOR FOR LOOP

```
SQL> set serveroutput on;
SQL> declare
 2 begin
 3 for emy in (select eid,ename from ssempp)
 4 loop
 5 dbms_output.put_line('Employee id and employee name are '|| emy.eid ‘and’||
emy.ename);
 6 end loop;
 7 end;
 8 /
Employee id and employee name are 1 and nala
Employee id and employee name are 2 and kala
Employee id and employee name are 5 and ajay
Employee id and employee name are 6 and vijay
Employee id and employee name are 3 and nila

PL/SQL procedure successfully completed.
```

TO WRITE A PL/SQL BLOCK TO UPDATE THE SALARY OF ALL EMPLOYEES

WHERE DEPARTMENT NO IS 11 BY 5000 USING CURSOR FOR LOOP AND TO DISPLAY THE UPDATED TABLE

```
SQL> set serveroutput on;
SQL> declare
 2 cursor cem is select eid,ename,sal,dno from ssempp where dno=11;
3 begin
 4 --open cem;
 5 for rem in cem
 6 loop
 7 update ssempp set sal=rem.sal+5000 where eid=rem.eid;
 8 end loop;
 9 --close cem;
 10 end;
 11 /

PL/SQL procedure successfully completed.

SQL> select * from ssempp;

 EID ENAME JOB SAL DNO
--------- ------------------- ------------------- --------- ---------
 1 nala lecturer 39000 11
 2 kala seniorlecturer 20000 12
 5 ajay lecturer 35000 11
 6 vijay lecturer 23000 11
 3 nila professor 60000 12
```

TO WRITE A PL/SQL BLOCK TO DISPLAY THE EMPLOYEE ID AND EMPLOYEE NAME WHERE DEPARTMENT NUMBER IS 11 USING EXPLICIT CURSORS

```
 1 declare
 2 cursor cenl is select eid,sal from ssempp where dno=11;
 3 ecode ssempp.eid%type;
 4 esal empp.sal%type;
 5 begin
 6 open cenl;
 7 loop
 8 fetch cenl into ecode,esal;
 9 exit when cenl%notfound;
 10 dbms_output.put_line(' Employee code and employee salary are' || ecode ‘and’||
esal);
 11 end loop;
 12 close cenl;
 13* end;
SQL> /
Employee code and employee salary are 1 and 39000
Employee code and employee salary are 5 and 35000
Employee code and employee salary are 6 and 23000

PL/SQL procedure successfully completed.
```

TO WRITE A PL/SQL BLOCK TO UPDATE THE SALARY BY 5000 WHERE THE JOB IS LECTURER , TO CHECK IF UPDATES ARE MADE USING IMPLICIT CURSORS AND TO DISPLAY THE UPDATED TABLE

```
SQL> declare
 2 county number;
 3 begin
 4 update ssempp set sal=sal+10000 where job='lecturer';
 5 county:= sql%rowcount;
 6 if county > 0 then
 7 dbms_output.put_line('The number of rows are '|| county);
 8 end if;
 9 if sql %found then
 10 dbms_output.put_line('Employee record modification successful');
 11 else if sql%notfound then
 12 dbms_output.put_line('Employee record is not found');
 13 end if;
 14 end if;
 15 end;
 16 /
The number of rows are 3

Employee record modification successful

PL/SQL procedure successfully completed.

SQL> select * from ssempp;

 EID ENAME JOB SAL DNO
--------- ------------------- ------------------- --------- ---------
 1 nala lecturer 44000 11
 2 kala seniorlecturer 20000 12
 5 ajay lecturer 40000 11
 6 vijay lecturer 28000 11
 3 nila professor 60000 12
```

RESULT

Thus the various operations were performed on the table using cursors and the output was verified.

PROCEDURES AND FUNCTIONS

EX NO: 11
DATE:

AIM

To write PL/SQL programs that execute the concept of functions and procedures.

DEFINITION

A procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task. They are essentially sub-programs. Procedures and functions are made up of,

- Declarative part
- Executable part
- Optional exception handling part

These procedures and functions do not show the errors.

KEYWORDS AND THEIR PURPOSES

REPLACE: It recreates the procedure if it already exists.
PROCEDURE: It is the name of the procedure to be created.
ARGUMENT: It is the name of the argument to the procedure. Paranthesis can be omitted if no arguments are present.
IN: Specifies that a value for the argument must be specified when calling the procedure ie. used to pass values to a sub-program. This is the default parameter.
OUT: Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie. used to return values to a caller of the sub-program.
INOUT: Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it's calling environment after execution.
RETURN: It is the datatype of the function's return value because every function must

return a value, this clause is required.

PROCEDURES – SYNTAX

```
 create or replace procedure <procedure name> (argument {in,out,inout} datatype )
{is,as}
 variable declaration;
 constant declaration;
 begin
 PL/SQL subprogram body;
 exception
 exception PL/SQL block;
 end;
```

FUNCTIONS – SYNTAX

```
 create or replace function <function name> (argument in datatype,……) return
datatype {is,as}
 variable declaration;
 constant declaration;
 begin
 PL/SQL subprogram body;
 exception
 exception PL/SQL block;
 end;
```

CREATING THE TABLE 'ITITEMS' AND DISPLAYING THE CONTENTS

```
SQL> create table ititems(itemid number(3), actualprice number(5), ordid number(4),
prodid number(4));

Table created.

SQL> insert into ititems values(101, 2000, 500, 201);

1 row created.

SQL> insert into ititems values(102, 3000, 1600, 202);

1 row created.

SQL> insert into ititems values(103, 4000, 600, 202);

1 row created.

SQL> select * from ititems;

ITEMID ACTUALPRICE ORDID PRODID
```

```
--------- ----------- -------- ---------
 101 2000 500 201
 102 3000 1600 202
 103 4000 600 202
```

PROGRAM FOR GENERAL PROCEDURE – SELECTED RECORD'S PRICE IS INCREMENTED BY 500 , EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE

```
SQL> create procedure itsum(identity number, total number) is price
number;  2 null_price exception;
 3 begin
 4 select actualprice into price from ititems where itemid=identity;
5 if price is null then
 6 raise null_price;
 7 else
 8 update ititems set actualprice=actualprice+total where itemid=identity;
9 end if;
 10 exception
 11 when null_price then
 12 dbms_output.put_line('price is null');
 13 end;
 14 /

Procedure created.

SQL> exec itsum(101, 500);

PL/SQL procedure successfully completed.

SQL> select * from ititems;

 ITEMID ACTUALPRICE ORDID PRODID
--------- ----------- --------- ---------
 101 2500 500 201
 102 3000 1600 202
 103 4000 600 202
```

PROCEDURE FOR 'IN' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure yyy (a IN number) is price
number;  2 begin
 3 select actualprice into price from ititems where itemid=a;
4 dbms_output.put_line('Actual price is ' || price);
 5 if price is null then
 6 dbms_output.put_line('price is null');
 7 end if;
 8 end;
 9 /

Procedure created.

SQL> exec yyy(103);
Actual price is 4000

PL/SQL procedure successfully completed.
```

PROCEDURE FOR 'OUT' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure zzz (a in number, b out number) is identity
number;  2 begin
 3 select ordid into identity from ititems where itemid=a;
4 if identity<1000 then
 5 b:=100;
 6 end if;
 7 end;
 8 /

Procedure created.

SQL> declare
 2 a number;
 3 b number;
 4 begin
 5 zzz(101,b);
 6 dbms_output.put_line('The value of b is '|| b);
 7 end;
 8 /
The value of b is 100

PL/SQL procedure successfully completed.
```

PROCEDURE FOR 'INOUT' PARAMETER – CREATION, EXECUTION

```
SQL> create procedure itit ( a in out number) is
 2 begin
 3 a:=a+1;
 4 end;
 5 /

Procedure created.
```

```
SQL> declare
 2 a number:=7;
 3 begin
 4 itit(a);
 5 dbms_output.put_line('The updated value is '||a);
 6 end;
 7 /

The updated value is 8
PL/SQL procedure successfully completed.
```

CREATE THE TABLE 'ITTRAIN' TO BE USED FOR FUNCTIONS

```
SQL>create table ittrain ( tno number(10), tfare number(10)); Table

created.

SQL>insert into ittrain values (1001, 550);

1 row created.

SQL>insert into ittrain values (1002, 600);

1 row created.

SQL>select * from ittrain;

 TNO TFARE
 --------- ------------
 1001 550
 1002 600
```

PROGRAM FOR FUNCTION AND IT'S EXECUTION

```
SQL> create function aaa (trainnumber number) return number is
2 trainfunction ittrain.tfare % type;
 3 begin
 4 select tfare into trainfunction from ittrain where tno=trainnumber;
5 return(trainfunction);
 6 end;
 7 /
Function created.

SQL> set serveroutput on;
SQL> declare
```

```
 2 total number;
 3 begin
 4 total:=aaa (1001);
 5 dbms_output.put_line('Train fare is Rs. '||total);
 6 end;
 7 /
Train fare is Rs.550

PL/SQL procedure successfully completed.
```

FACTORIAL OF A NUMBER USING FUNCTION — PROGRAM AND EXECUTION

```
 SQL> create function itfact (a number) return number is
 2 fact number:=1;
 3 b number;
 4 begin
 5 b:=a;
 6 while b>0
 7 loop
 8 fact:=fact*b;
 9 b:=b-1;
 10 end loop;
 11 return(fact);
 12 end;
 13 /

Function created.

 SQL> set serveroutput on;
SQL> declare
 2 a number:=7;
 3 f number(10);
 4 begin
 5 f:=itfact(a);
 6 dbms_output.put_line('The factorial of the given number is'||f);
7 end;
 8 /

The factorial of the given number is 5040

PL/SQL procedure successfully completed.
```

RESULT

Thus the PL/SQL programs were executed and their respective outputs were verified.

EMBEDDED SQL

EX NO: 12
DATE:

AIM:

To execute the embedded SQL program in JAVA.

CODE:

```
import java.sql.*;
class emb
{
public static void main(String args[]) throws
Exception {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String s="insert into table1 values ("+args[0]+")";
Connection con =
DriverManager.getConnection("jdbc:odbc:aarthi"); Statement
st=con.createStatement();
int i =st.executeUpdate(s);
if(i>0)
System.out.println("Data Inserted" +i);
else
System.out.println("Data not inserted");
con.close();
}
}
```

OUTPUT:

Table before insertion:

Table1
Id

```
D:\Java\jdk1.5.0\bin>javac embedded1.java
D:\Java\jdk1.5.0\bin>java embedded1 1001
Data Inserted1

D:\Java\jdk1.5.0\bin>java embedded1 1002
Data Inserted1

D:\Java\jdk1.5.0\bin>java embedded1 1003
Data Inserted1

D:\Java\jdk1.5.0\bin>java embedded1 1004
Data Inserted1

D:\Java\jdk1.5.0\bin>
```

Table after insertion:

Table1
Id
1001
1001
1002
1003
1004

RESULT:

Thus the embedded SQL application is implemented successfully.

Application Development using appropriate Front End and Back End Tools

EX NO: 13

DATE:

AIM:

To develop a JAVA application for employee information system.

USING MySQL:

ALGORITHM:

```
Step 1: Start.
Step 2: Import required packages.
Step 3: Invoke the database driver and create a connection to the database.
Step 4: Display the list of all operations and get the user’s choice.
Step 5: If the choice is 1, display the whole contents of the table.
Step 6: If the choice is 2, get the required data from the user and add a new recordto the database.
Step 7: If the choice is 3, get the eid of the required employee and delete the record from the table.
Step 8: If the choice is 4, ask the user as to which column must be updated.
Step 8i: If the choice is 1, list the options for salary updation and get the user’s choice.
        Step 8ii: If the choice is 1, get the increment amount and increment the salaryof all the employees.
        Step 8iii: If the choice is 2, get the eid and the new salary of the employee and update the salary.
    Step 8iv: If the choice 2, get the new bonus for all the employees and update itin the database.
    Step 8v: If the choice is 3, get the eid and the new role of the employee and update it in the database.
Step 9: If the choice is 5, display the total count of employees in the database.
Step 10: If the choice is 6, get the eid of the employee, calculate the pay including the bonus and print it.
Step 11: Repeat steps 4-10, if the user wants to continue.
Step 12: Close the connection to the database.
Step 13: End.
```

PROGRAM:

a) For accessing table in MySQL:

Table Creation:

```
create table employee_db(
eid int,
ename varchar(20),
salary int,
bonus int,
role varchar(20));
```

JDBC Program:

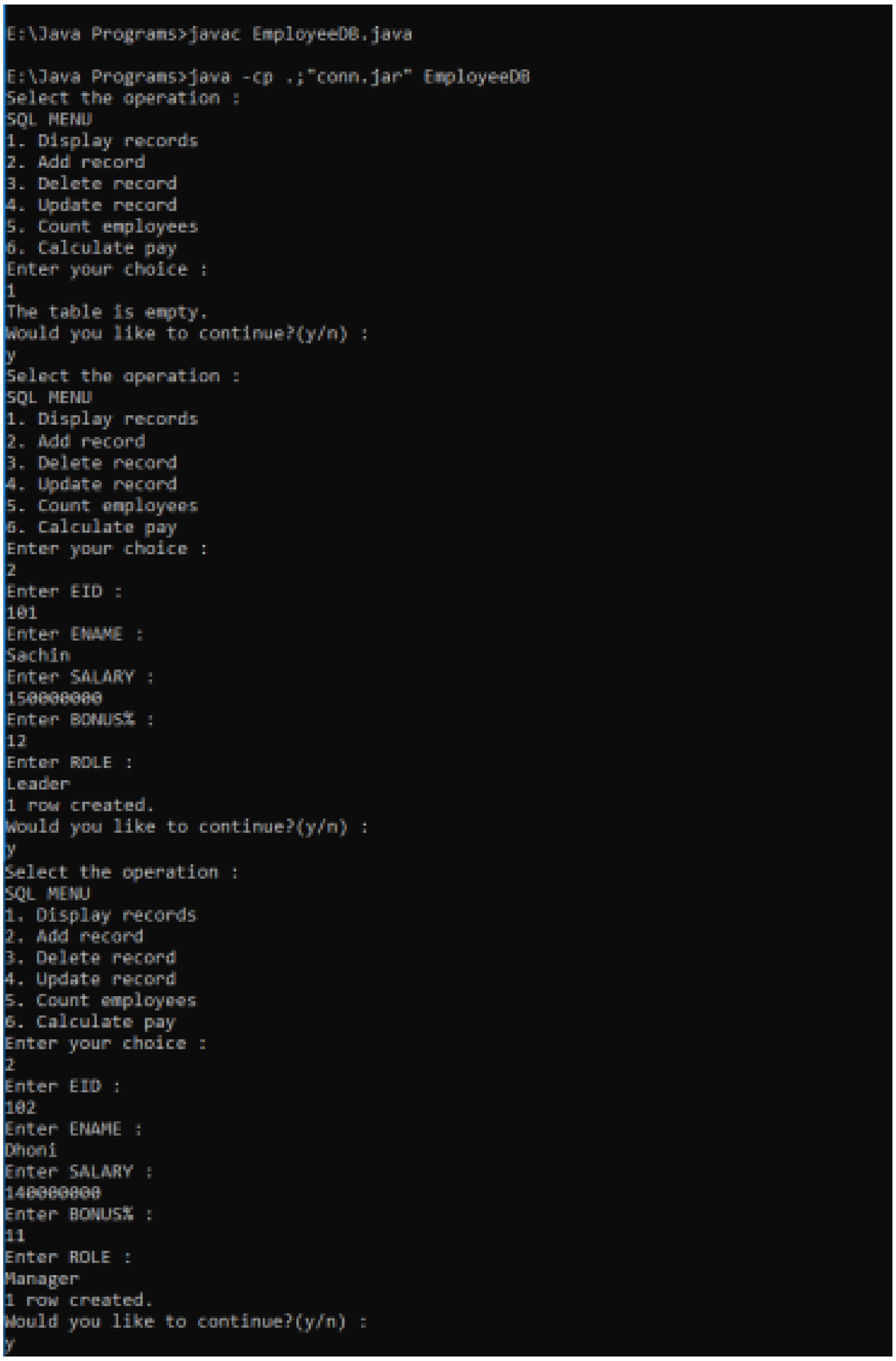
```
import java.sql.*;
import java.util.Scanner;
public class EmployeeDB
{
        public void display(Statement stmt)
        {
                try
                {
                        String q="select * from employee_db";
                        ResultSet r=stmt.executeQuery(q);
                        int i=0;
                        if(r.next()==false)
                        {
                                System.out.println("The table is empty.");
                                return;
                        }
                        do
                        {
                                System.out.println("Row "+(++i)+" : ");
                                        System.out.println("EID : "+r.getString("EID"));
                                System.out.println("ENAME : "+r.getString("ENAME"));
                                System.out.println("SALARY : "+r.getString("SALARY"));
                                System.out.println("BONUS : "+r.getString("BONUS"));
                                    System.out.println("ROLE : "+r.getString("ROLE"));
                                System.out.println();
                        }while(r.next());
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
        public void add(Statement stmt,Scanner s)
        {
                try
                {
                        System.out.println("Enter EID : ");int
                        eid=s.nextInt();
                        s.nextLine(); System.out.println("Enter
                        ENAME : "); String ename=s.nextLine();
                        System.out.println("Enter SALARY : ");
                int salary=s.nextInt();
                s.nextLine(); System.out.println("Enter
                BONUS% : ");int bonus=s.nextInt();
                s.nextLine(); System.out.println("Enter
                ROLE : ");String role=s.nextLine();
                String q=String.format("insert into employee_db
 values(%d,'%s',%d, %d,'%s')",eid,ename,salary,bonus,role);
                int status=stmt.executeUpdate(q);
                if(status==1)
                     System.out.println("1 row created.");
                 else
                     System.out.println("Couldn't create the row!");
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
        public void delete(Statement stmt,Scanner s)
        {
            try
                {System.out.println("Enter the EID of the record to be deleted : ");

                int eid=s.nextInt();
                s.nextLine();
                String q=String.format("delete from employee_db where
                eid=%d",eid); int status=stmt.executeUpdate(q);
                if(status==1)
                     System.out.println("1 row deleted.");
                 else
                     System.out.println("Couldn't delete the row!");
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
public void update(Statement stmt,Scanner s)
        {
            try
            {
                String q; int
                status;
                System.out.println("Select the column to be updated : ");
                System.out.println("1. SALARY");
                System.out.println("2. BONUS");
                System.out.println("3. ROLE");
                System.out.println("Enter your choice : ");int
                ch=s.nextInt();
                s.nextLine();
                if(ch==1)
                {
                     System.out.println("SALARY UPDATE MENU");
                      System.out.println("1. Increment salary for everyone");
                     System.out.println("2. Update a new salary for an employee");
                     System.out.println("Enter your choice : ");
                     int c=s.nextInt();
                     s.nextLine();
                     if(c==1)
                     {
                          System.out.println("Enter the increment amount : ");
                          int incr=s.nextInt();
                          s.nextLine();
                          q=String.format("update employee_db set salary=salary+%d",
incr);                                   System.out.println("1 row
                                         updated"); else if(status>1)
                                         System.out.println(status+" rows
                                         updated."); else
                                         System.out.println("Couldn't
                                         update!");


}
status=stmt.executeUpdate(q);
if(status==1)
                     else if(c==2)
                     {
                       System.out.println("Enter the EID of the employee : ");
                       int eid=s.nextInt();
                       s.nextLine();
                       System.out.println("Enter the new salary : ");
                       int sal=s.nextInt();
                       s.nextLine();
                       q=String.format("update employee_db set SALARY=%d where
                       EID = %d",sal,eid);
```

```
                    status=stmt.executeUpdate(q);
                    if(status!=0)
                            System.out.println("1 row updated ");
                    else
                            System.out.println("Couldn’t update! ");
}
else
        System.out.println("Invalid Command!");
                }
                else if(ch==2)
                {
                     System.out.println("Enter the new bonus for all employees : ");
                     int b=s.nextInt();
                     s.nextLine();
                     q=String.format("update employee_db set bonus=%d",b);
                     status=stmt.executeUpdate(q);
                     if(status==1)
                          System.out.println("1 row updated.");
                     else if(status>1)
                          System.out.println(status+" rows updated.");
                     else
                          System.out.println("Couldn't update!");
                }
                else if(ch==3)
                {
                        System.out.println("Enter the EID of the employee : ");
                        int eid=s.nextInt();
                        s.nextLine();
                        System.out.println("Enter the new ROLE : ");
                        String role=s.nextLine();
                         q=String.format("update employee_db set role='%s' where
                eid=%d",role,id);
                        status=stmt.executeUpdate(q);
                        if(status!=0)
                                System.out.println(status+" row updated");
                        else
                                System.out.println("Couldn't update!");
                }
                else
                        System.out.println("Invalid Command!");

        catch(Exception e)
        {
```

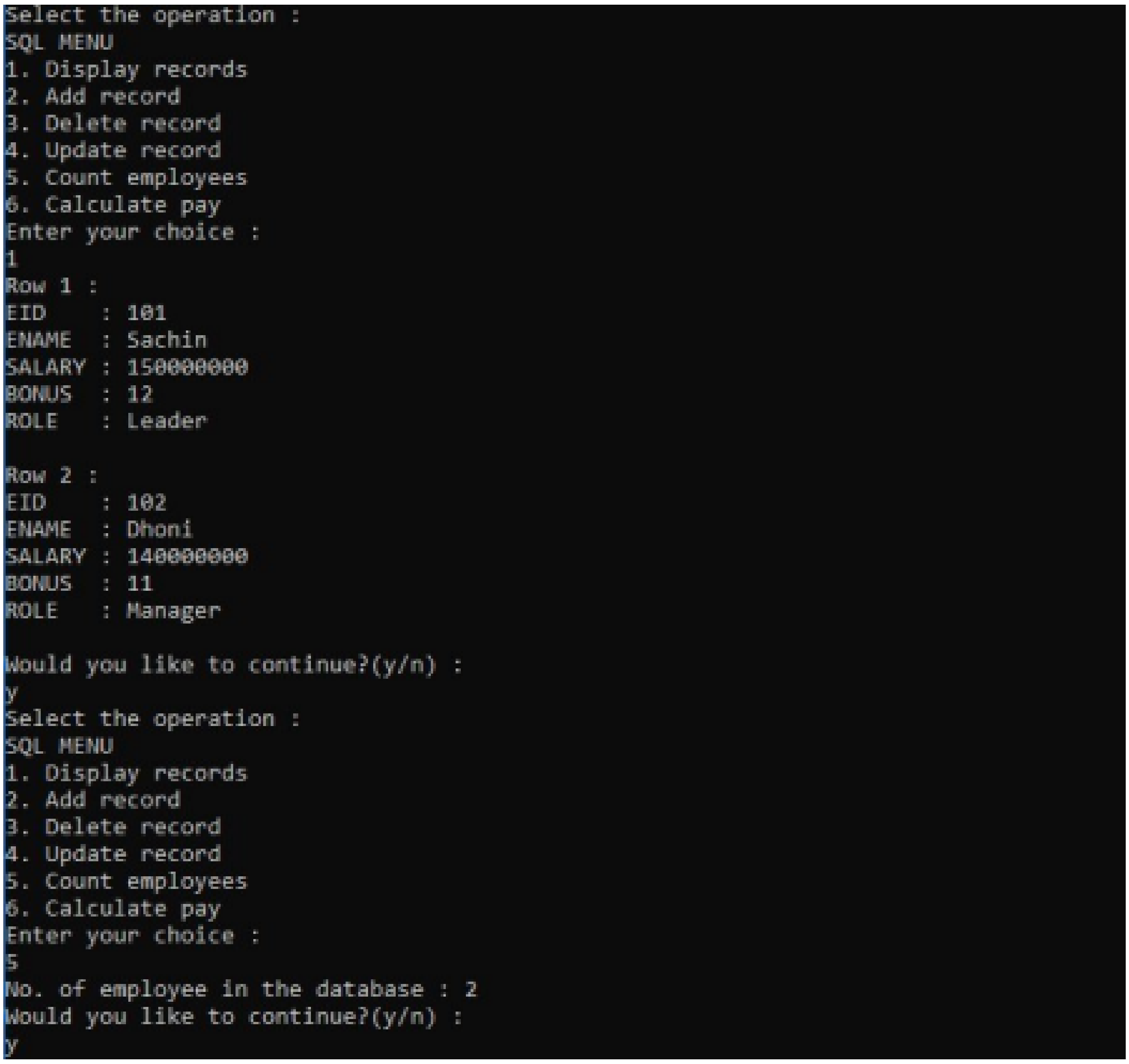
```
                    System.out.println(e);
          }
}
public void count(Statement stmt)
{
try

{
          String q="select count(*) from employee_db";
          ResultSet r=stmt.executeQuery(q);
          r.next();
          System.out.println("No. of employee in the database :
"+r.getString("COUNT(*)"));
}
catch(Exception e)
{
          System.out.println(e);
}
}
public void pay(Statement stmt,Scanner s)
{
 try
{
          System.out.println("Enter the EID of the employee : ");
           int eid=s.nextInt();
          s.nextLine();
          String q=String.format("select salary,bonus from employee_db where
eid=%d",eid);
          ResultSet r=stmt.executeQuery(q); r.next();
          int sal=r.getInt("SALARY"); int bon=r.getInt("BONUS"); double
pay=(1+(bon/100.0))*sal;
          System.out.println("Employee's pay : "+pay);
}
catch(Exception e)
{
          System.out.println(e);
}
}
public static void main(String[] args)
{
          Scanner s=new Scanner(System.in); try
{
Class.forName("com.mysql.cj.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","root");
char c;
```

```
do
{
          System.out.println("Select the operation : ");
          System.out.println("SQL MENU");
          System.out.println("1. Display records");
          System.out.println("2. Add record");
           System.out.println("3. Delete record");
          System.out.println("4. Update record");
          System.out.println("5. Count employees");
          System.out.println("6. Calculate pay");
          System.out.println("Enter your choice : ");
          int ch=s.nextInt();
          s.nextLine();
          EmployeeDB e=new EmployeeDB();
          Statement stmt=con.createStatement();
          switch(ch)
          {
          case 1: e.display(stmt);
                    break;
          case 2: e.add(stmt,s);
                    break;
          case 3: e.delete(stmt,s);
                    break;
          case 4: e.update(stmt,s);
                    break;
          case 5: e.count(stmt);
                    break;
          case 6: e.pay(stmt,s);
                    break;
          default:System.out.println("Invalid Command!");
          }
System.out.println("Would you like to continue?(y/n) : ");
c=s.nextLine().charAt(0);
}while(c=='y'||c=='Y');
con.close();
 }
 catch(Exception e)
 {
 System.out.println(e);
 }
 s.close();
 }
 }
```

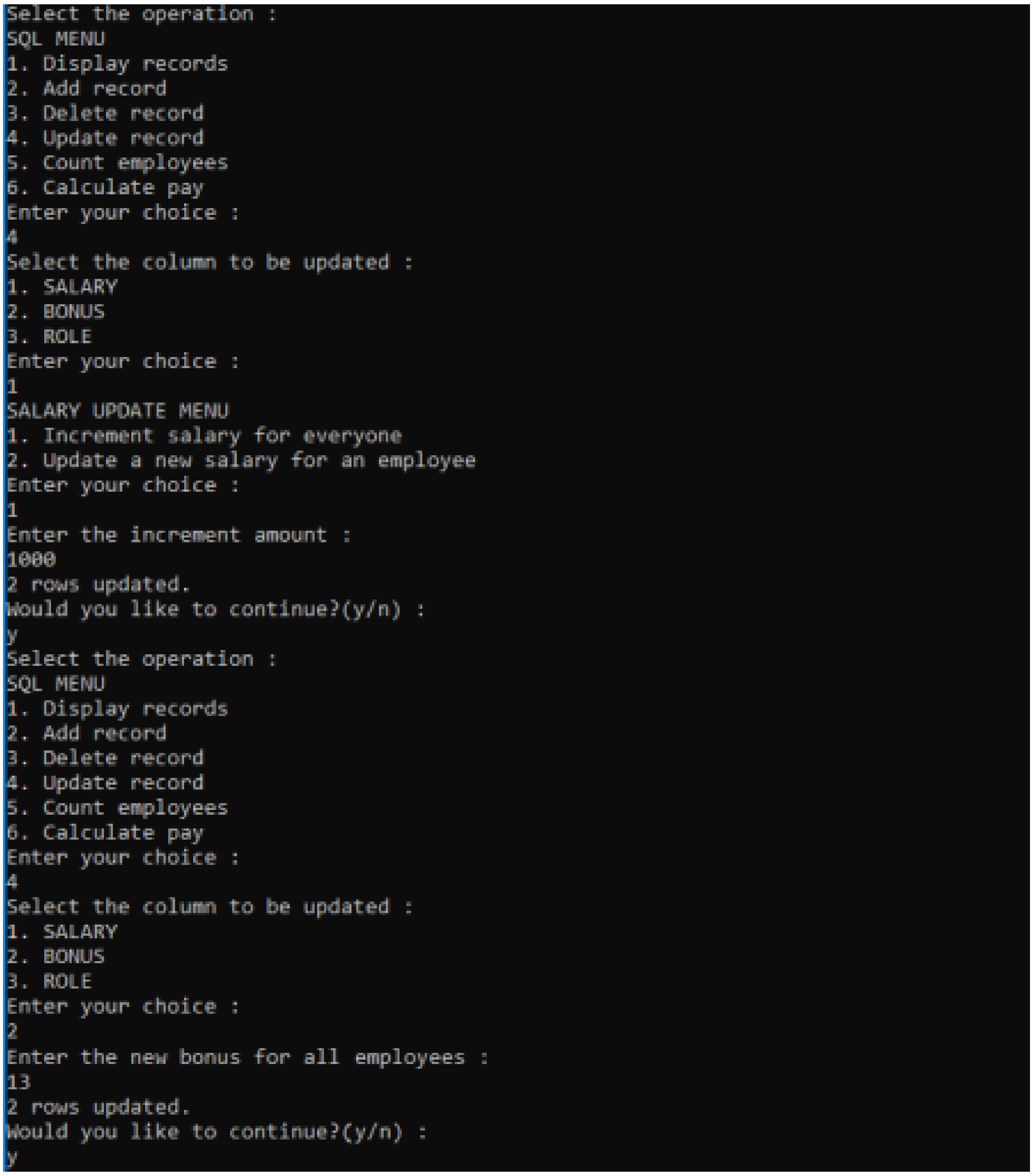
OUTPUT:



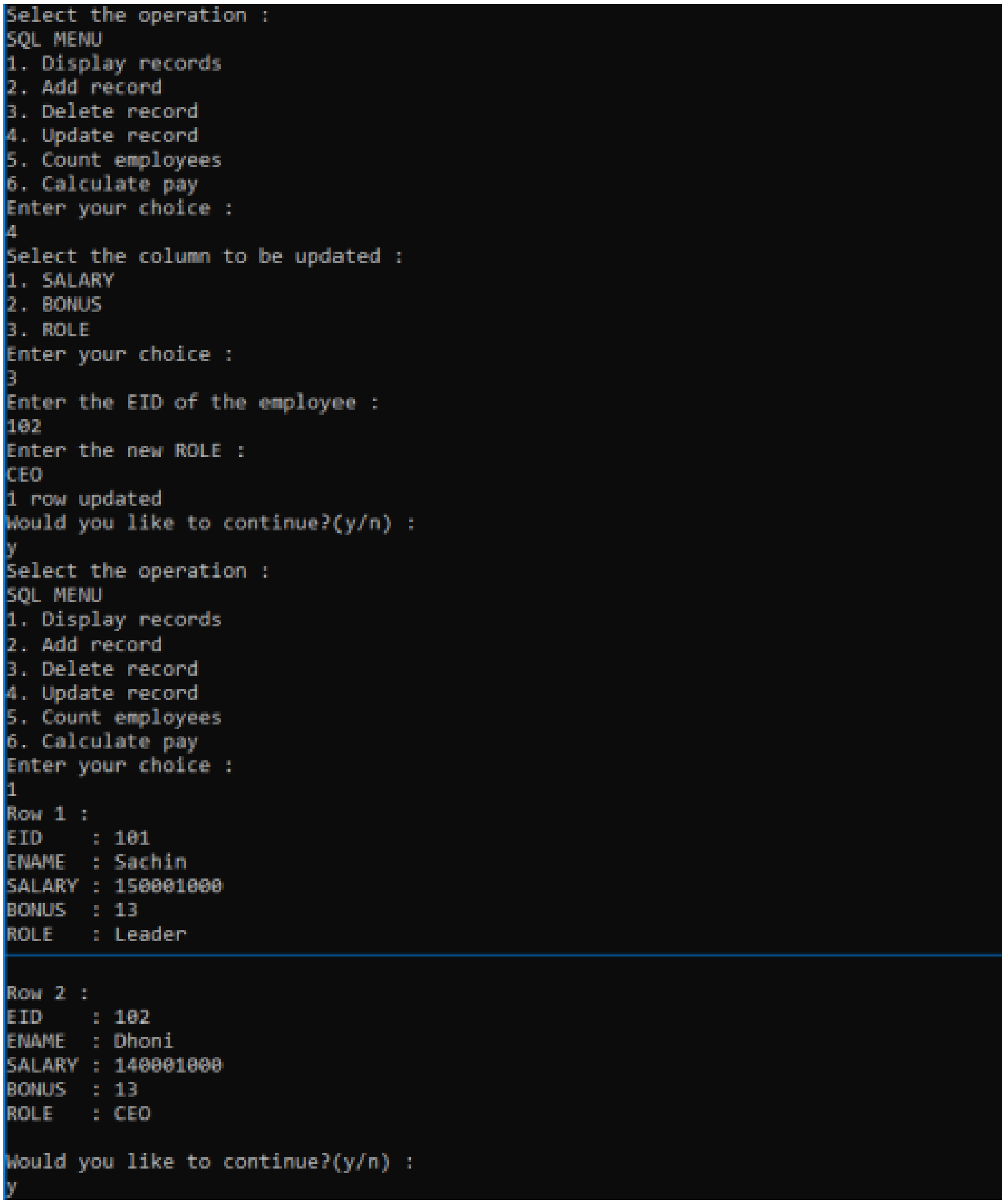
```
E:\Java Programs>javac EmployeeDB.java

E:\Java Programs>java -cp .;"conn.jar" EmployeeDB
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
1
The table is empty.
Would you like to continue?(y/n) :
y
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
2
Enter EID :
101
Enter ENAME :
Sachin
Enter SALARY :
150000000
Enter BONUS% :
12
Enter ROLE :
Leader
1 row created.
Would you like to continue?(y/n) :
y
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
2
Enter EID :
102
Enter ENAME :
Dhoni
Enter SALARY :
140000000
Enter BONUS% :
11
Enter ROLE :
Manager
1 row created.
Would you like to continue?(y/n) :
y
```



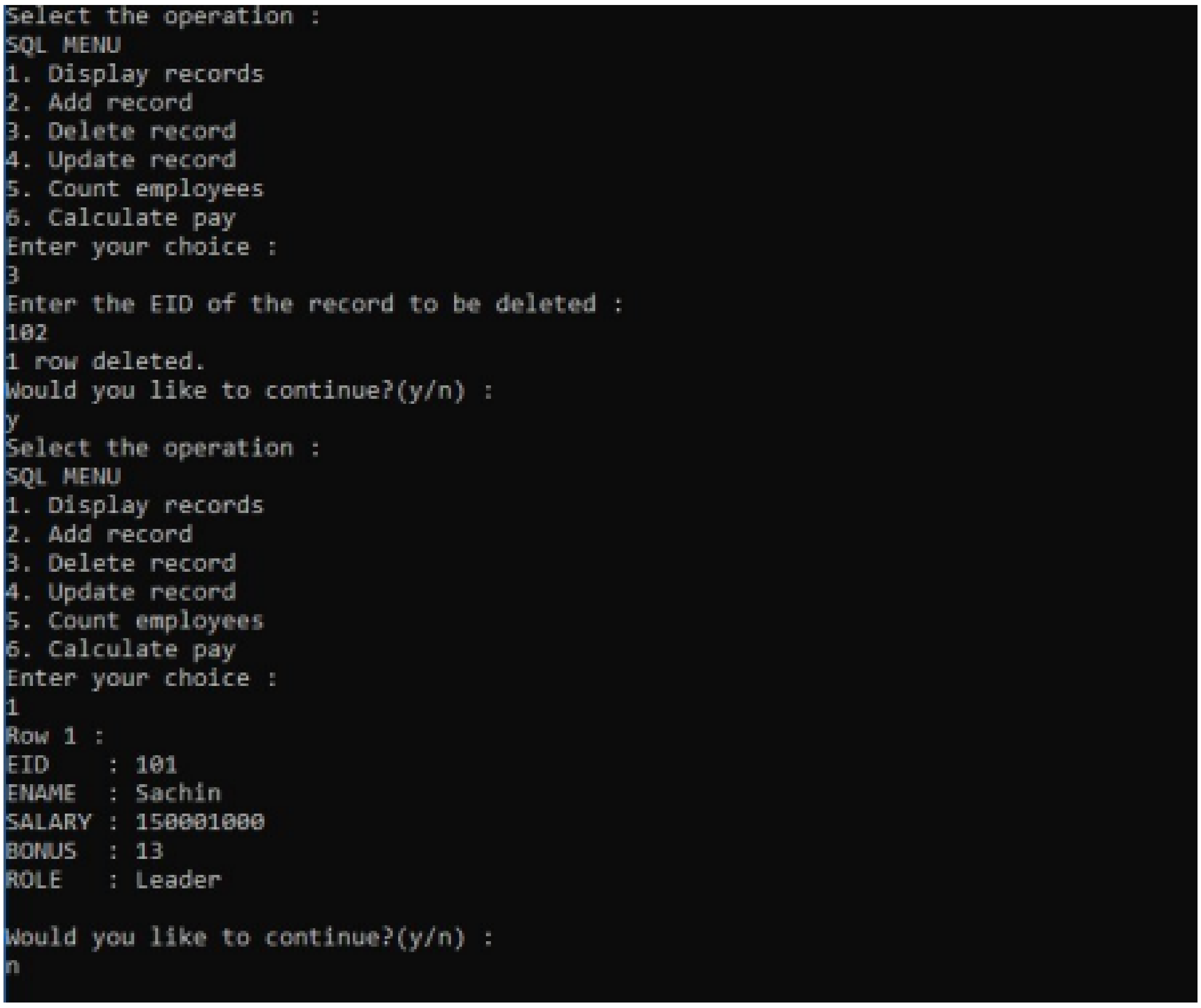
```
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
1
Row 1 :
EID    : 101
ENAME  : Sachin
SALARY : 150000000
BONUS  : 12
ROLE   : Leader

Row 2 :
EID    : 102
ENAME  : Dhoni
SALARY : 140000000
BONUS  : 11
ROLE   : Manager

Would you like to continue?(y/n) :
y
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
5
No. of employee in the database : 2
Would you like to continue?(y/n) :
y
```



```
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
4
Select the column to be updated :
1. SALARY
2. BONUS
3. ROLE
Enter your choice :
1
SALARY UPDATE MENU
1. Increment salary for everyone
2. Update a new salary for an employee
Enter your choice :
1
Enter the increment amount :
1000
2 rows updated.
Would you like to continue?(y/n) :
y
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
4
Select the column to be updated :
1. SALARY
2. BONUS
3. ROLE
Enter your choice :
2
Enter the new bonus for all employees :
13
2 rows updated.
Would you like to continue?(y/n) :
y
```



```
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
4
Select the column to be updated :
1. SALARY
2. BONUS
3. ROLE
Enter your choice :
3
Enter the EID of the employee :
102
Enter the new ROLE :
CEO
1 row updated
Would you like to continue?(y/n) :
y
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
1
Row 1 :
EID    : 101
ENAME  : Sachin
SALARY : 150001000
BONUS  : 13
ROLE   : Leader

Row 2 :
EID    : 102
ENAME  : Dhoni
SALARY : 140001000
BONUS  : 13
ROLE   : CEO

Would you like to continue?(y/n) :
y
```



```
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
3
Enter the EID of the record to be deleted :
102
1 row deleted.
Would you like to continue?(y/n) :
y
Select the operation :
SQL MENU
1. Display records
2. Add record
3. Delete record
4. Update record
5. Count employees
6. Calculate pay
Enter your choice :
1
Row 1 :
EID    : 101
ENAME  : Sachin
SALARY : 150001000
BONUS  : 13
ROLE   : Leader

Would you like to continue?(y/n) :
n
```

RESULT:

The JAVA application for employee information system was implemented and executed successfully.