CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

Department Of Computer Science And Engineering

Year:2nd **Semester:4th**

20 CS C14 – Database Management Systems Lab

LAB MANUAL

INDEX

S.No	Practical's Name	Date	Remark
1	Write the queries for Data Manipulation and Data Definition Language.	17-12	
2	Write SQL queries for student and employee database.	24-12	
3	Write SQL query using using character and numeric functions.	31-12	
4	Write SQL queries using general functions and conversion functions.	21-01	
5	Write SQL queries using constraints,joins and group functions.	04-02	
6	Write SQL queries to demonstrate views.	11-02	
7	Write programs by the use of PL/SQL	18-02	
8	Write programs on procedures and functions	25-02	
9	Write programs on cursors	11-03	
10	Write programs on exception handling and triggers	18-03	

AIM: Write the queries for Data Manipulation and Data Definition Language.

THEORY:

DDL: A **data definition language** or **data description language** (**DDL**) is syntax similar to a computer programming language for defining data structures, especially database schemas.-

Commands in DDL are:

a. CREATE
b. DROP
c. RENAME
d. ALTER

DDL COMMANDS:

SYNTAX:

CREATE Statement: *CREATE TABLE tablename (attribute_1 data type, attribute_2 data type, …… attribute_n data type);*

DROP Statement:*DROP TABLE table_name;*

RENAME Statement:*RENAME table_name to new_name;*

ALTER Statement:
Add column to Table: *ALTER TABLE table_name ADD column_name column-definition*;
Modify column in Table: *ALTER TABLE table_name MODIFY column_name column_type;*
Drop column in Table*: ALTER TABLE table_name DROP COLUMN column_name;*

DDL QUERIES:

Q1.Write a query to create a table student1 with sid, sname, and ssec.

```
create table student1
( sid varchar(10),sname varchar(20),ssec varchar(4) );
```

Table Created

.

Q2. Write a query to modify the datatype of sid to varchar.

```
Alter table student1
   modify sid char(10);
```

Table altered

Q3. Write a query to add column age to the table student1.

Alter table student1

Add age Number

Table altered.

Q4. Write a query to drop the column age from student1.

Alter table student1

drop column age;

Table altered

Q5. Write a query to rename the table to Lab1.

Rename student1 to Lab1

Table renamed

Q6. Write a Query to drop the emp

DROP table emp;

Table deleted.

THEORY:

DML: A **data manipulation language** (**DML**) is a family of syntax elements similar to a computer programming language used for selecting, inserting, deleting and updating data in a database. Performing read-only queries of data is sometimes also considered a component of DML. Commands in DML are:

a. INSERT
b. UPDATE
c. DELETE
d. SELECT

DML COMMANDS:

SYNTAX:

INSERT Statement:
Single Row into a Table: INSERT INTO table – name [column- identifier-comma-list)]
VALUES (column-valuecomma-list);
Multiple Row into a Table: insert into <table name> values (&col1, &col2, ….);

UPDATE Statement: *UPDATE table-name SET update- column-list [WHERE search- condition];*

DELETE Statement: *DELETE FROM table-name [WHERE search- condition];*

DML QUERIES:

Q1.Write a query to insert values into table grade_report.

Sql->

```
INSERT INTO grade_report values(17,112,'B');

INSERT INTO grade_report values(17,119,'C');

INSERT INTO grade_report values(8,85,'C');

INSERT INTO grade_report values(8,92,'A');

INSERT INTO grade_report values(8,102,'B');

INSERT INTO grade_report values(8,135,'A');
```

Q2. Write a query to display values from table grade_report.

Sql->

```
SELECT * FROM grade_report;
```

	STUDENT_NUMBER	SECTION_IDENTIFIER	GRADE
1	17	112	B
2	17	119	C
3	8	85	C
4	8	92	A
5	8	102	B
6	8	135	A

Q3.Write a query to insert values to table student1 using substitution operator and display the table

Sql->

```
insert into Lab1(sid,sname,ssec)

values('&sid','&sname','&ssec');

Old2:values('&sid','&sname','&ssec')
new2:values('128','Shwetha','C-3')

select * from student1;
```

SID	SNAME	SSEC
129	Jyotika	C-2
130	Keerthana	C-2
131	Kranthi	C-2
132	Juhitha	C-3
133	Susmitha	C-3
134	Alekya	C-3
128	Shwetha	C-3

Q4.Write a query to update values, set sid to 106

Sql-> update studentsql

set sid=106,sbranch='mech'

where sid=101;

```
1 rows updated.
```

Q5.Write SQL queries using logical operations and operators.

select * from emp

where empsalary like '%00';

	EMPNAME	EMPID	EMPSALARY	DESG
1	Arun	100	50000	Delivery manager
2	Pooja	101	60000	Sales manager
3	Jahnavi	102	63000	Pre Sales manager
4	Dev	103	63000	HR

select * from emp

where empname LIKE 'Pooja';

	EMPNAME	EMPID	EMPSALARY	DESG
1	Pooja	101	60000	Sales manager

select empname,empid from emp

where empsalary>'50000' AND empsalary<='60000';

	EMPNAME	EMPID
1	Pooja	101

select empname from emp

where empsalary is NOT NULL;

	EMPNAME
1	Arun
2	Pooja
3	Jahnavi
4	Dev

select empname,empid from emp

where empsalary>'50000' OR empname='Dev';

	EMPNAME	EMPID
1	Pooja	101
2	Jahnavi	102
3	Dev	103

select * from emp

where empsalary >'60000';

	EMPNAME	EMPID	EMPSALARY	DESG
1	Jahnavi	102	63000	Pre Sales manager
2	Dev	103	63000	HR

select empname,empid from emp

where empsalary like '_0000';

	EMPNAME	EMPID
1	Arun	100
2	Pooja	101

AIM: Write the queries for evaluating single row functions.

THEORY:

Single row functions are the one who work on single row and return one output per row. For example, length and case conversion functions are single row functions. Single row functions can be character functions, numeric functions, date functions, and conversion functions. Note that these functions are used to manipulate data items. These functions require one or more input arguments and operate on each row, thereby returning one output value for each row. Argument can be a

column, literal or an expression. Single row functions can be used in SELECT statement, WHERE and ORDER BY clause.

Single row functions are of five types:

1. Character Functions
2. Number Functions
3. Date Functions
4. Conversion functions
5. General Functions

Character Functions:

These are subdivided into – Case Manipulation Functions and Character Manipulation Functions.

Case Manipulation Functions - Accepts character input and returns a character value. Functions under the category are UPPER, LOWER and INITCAP.

- UPPER function converts a string to upper case.
- LOWER function converts a string to lower case.
- INITCAP function converts only the initial alphabets of a string to upper case.

Character Manipulation Functions – Accepts character input and returns number or character value. Functions under the category are CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM and REPLACE.

- CONCAT function concatenates two string values.
- LENGTH function returns the length of the input string.
- SUBSTR function returns a portion of a string from a given start point to an end point.
- INSTR function returns numeric position of a character or a string in a given string.
- LPAD and RPAD functions pad the given string upto a specific length with a given character.
- TRIM function trims the string input from the start or end.
- REPLACE function replaces characters from the input string with a given character.

Number Functions :

Accepts numeric input and returns numeric values. Functions under the category are ROUND, TRUNC, and MOD.

- ROUND and TRUNC functions are used to round and truncate the number value.
- MOD is used to return the remainder of the division operation between two numbers.

Date Functions :

Date arithmetic operations return date or numeric values. Functions under the category are MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND and TRUNC.

- MONTHS_BETWEEN function returns the count of months between the two dates.
- ADD_MONTHS function add 'n' number of months to an input date.
- NEXT_DAY function returns the next day of the date specified.
- LAST_DAY function returns last day of the month of the input date.

- ROUND and TRUNC functions are used to round and truncates the date value.

Conversion Functions :

Type conversion can be either implicitly done by Oracle or explicitly done by the programmer.

Implicit Type Conversion – A VARCHAR2 or CHAR value can be implicitly converted to NUMBER or DATE type value by Oracle. Similarly, a NUMBER or DATA type value can be automatically converted to character data by Oracle server. Note that the impicit interconversion happens only when the character represents the a valid number or date type value respectively.

Explicit Type Conversion – SQL Conversion functions are single row functions which are capable of typecasting column value, literal or an expression . TO_CHAR, TO_NUMBER and TO_DATE are the three functions which perform cross modification of data types.

- TO_CHAR function is used to typecast a numeric or date input to character type with a format model (optional). Dates can be formatted in multiple formats after converting to character types using TO_CHAR function.
  Example Formats for dates :

Format Model	Description
YYYY	Full year in number
Year	Year spelled out
MM	Two digit value for month
Month	Full name of month
DD	Numeric day of month
DAY	Full name of day the week
TH	Produces ordinal number
SP	Spell out the number.
SPTH	Spell out the ordinal number.

Example formats for Numbers:

Format Model	Description
,	It returns a comma in the specified position. You can specify multiple commas in a number format model.

.	Returns a decimal point, in the specified position.
$	Returns value with a leading dollar sign
0	Returns leading zeros.
9	Returns value with the specified number of digits
L	It is used for local currency symbol

- The TO_NUMBER function converts a character value to a numeric datatype. If the string being converted contains nonnumeric characters, the function returns an error.
- The function takes character values as input and returns formatted date equivalent of the same. The TO_DATE function allows users to enter a date in any format, and then it converts the entry into the default format used by Oracle server.

General Functions :

General functions are used to handle NULL values in database. The objective of the general NULL handling functions is to replace the NULL values with an alternate value. We shall briefly see through these functions below.

- NVL – The NVL function substitutes an alternate value for a NULL value. NVL(arg, replace_with)
- NVL2 – NVL2 function can be used to substitute an alternate value for NULL as well as non NULL value. NVL2(arg, value_if_not_null,value_if_null);
- NULLIF – The NULLIF function compares two arguments expr1 and expr2. If expr1 and expr2 are equal, it returns NULL; else, it returns expr1. Unlike the other null handling function, first argument can't be NULL. NULLIF(exp1,exp2)
- COALESCE function, a more generic form of NVL, returns the first non-null expression in the argument list. It takes minimum two mandatory parameters but maximum arguments has no limits. COALESCE (exp1,exp2,.....,expn)

QUERIES:

Q1.Write queries to demonstrate all the case manipulation functions.

Select lower(major) from STUDENT;

	LOWER(MAJOR)
1	cs
2	cs
3	math
4	cs
5	math

select upper(name1) from STUDENT;

	UPPER(NAME1)
1	JUHITHA
2	KRANTHI
3	SUSMITHA
4	BROWN
5	JOHNSON

select initcap(name1) from STUDENT;

	INITCAP(NAME1)
1	Juhitha
2	Kranthi
3	Susmitha
4	Brown
5	Johnson

select lower(major)||initcap(name1) as Details from student ;

	DETAILS
1	csJuhitha
2	csKranthi
3	mathSusmitha
4	csBrown
5	mathJohnson

select name1,student_number,class1 from student

where upper(major)='CS';

	NAME1	STUDENT_NUMBER	CLASS1
1	Juhitha	28	2
2	Kranthi	31	2
3	Brown	8	2

Q2.Write queries to demonstrate all the character manipulation functions.

select concat(student_number,name1) from student;

	CONCAT(STUDENT_NUMBER,NAME1)
1	28Juhitha
2	31Kranthi
3	42Susmitha
4	8Brown
5	25Johnson

select substr(name1,1,4) from student;

	SUBSTR(NAME...
1	Juhi
2	Kran
3	Susm
4	Brow
5	John

select substr(name1,4) from student;

	SUBSTR(NAME1,4)
1	itha
2	nthi
3	mitha
4	wn
5	nson

select length(student_number) from student;

	LENGTH(STUDENT_NUMBER)
1	2
2	2
3	2
4	1
5	2

select length('Hello WORLd') from dual;

	LENGTH('HELLOWORLD')
1	11

select instr(name1,'a') from student;

	INSTR(NAME1,'A')
1	7
2	3
3	8
4	0
5	0

select instr(name1,'a',1,2) from student;

	INSTR(NAME1,'A',1,2)
1	0
2	0
3	0
4	0
5	0

select lpad(section_identifier,10,'-') from section1;

	LPAD(SECTION_IDENTIFIER,10,'-')
1	--------85
2	--------92
3	-------102
4	-------112
5	-------119
6	-------135

select rpad(section_identifier,10,'*') from section1;

	RPAD(SECTION_IDENTIFIER,10,'*')
1	85********
2	92********
3	102*******
4	112*******
5	119*******
6	135*******

select trim('G'from instuctor) from section1;

	TRIM('G'FROMINSTUCTOR)
1	KIN
2	ANDER
3	KNUTH
4	CHAN
5	ANDER
6	STONE

SELECT TRIM(LEADING 'H' FROM 'HHIIHH') FROM DUAL;

	TRIM(LEADING'H'FROM'HHIIHH')
1	IIHH

select replace('Mississipie','s','x') from dual;

	REPLACE('MISSISSIPIE','S','X')
1	Mixxixxipie

SELECT CONCAT(name1,concat('-',student_number)) from student;

	CONCAT(NAME1,CONCAT('-',STUDENT_NUMBER))
1	Juhitha-28
2	Kranthi-31
3	Susmitha-42
4	Brown-8
5	Johnson-25

Q3.Write a query to demonstrate all the number functions.

SELECT ROUND(45.89112) FROM DUAL;

	ROUND(45.89112)
1	46

SELECT ROUND(45.89112,1) FROM DUAL;

	ROUND(45.89112,1)
1	45.9

SELECT ROUND(45.89112,2) FROM DUAL;

	ROUND(45.89112,2)
1	45.89

SELECT TRUNC(45.89112) FROM DUAL;

	TRUNC(45.89112)
1	45

SELECT TRUNC(45.89112,1) FROM DUAL;

	TRUNC(45.89112,1)
1	45.8

SELECT MOD(1002,55) FROM DUAL;

	MOD(1002,55)
1	12

Q4.Write queries to demonstrate all the date functions.

SELECT SYSDATE FROM DUAL;

	SYSDATE
1	31-DEC-18

SELECT MONTHS_BETWEEN(SYSDATE,'12-NOV-2018') FROM DUAL;

	MONTHS_BETWEEN(SYSDATE,'12-NOV-2018')
1	1.632862529868578255675029868578255675 03

SELECT ADD_MONTHS(SYSDATE,'4') FROM DUAL;

	ADD_MONTHS(SYSDA...
1	30-APR-19

SELECT NEXT_DAY('21-FEB-18','MONDAY') FROM DUAL;

	NEXT_DAY('21-FEB-18','MONDAY')
1	26-FEB-18

SELECT LAST_DAY('21-FEB-18') FROM DUAL;

	LAST_DAY('21-FEB-18')
1	28-FEB-18

SELECT ROUND(TO_DATE('19-DEC-2017'),'MONTH') FROM DUAL;

	ROUND(TO_DATE('19-DEC-2017'),'MONTH')
1	01-JAN-18

SELECT TRUNC(TO_DATE('19-DEC-2017'),'YEAR') FROM DUAL;

	TRUNC(TO_DATE('19-DEC-2017'),'YEAR')
1	01-JAN-17

SELECT ROUND(TO_DATE('19-MAR-2017'),'YEAR') FROM DUAL;

	ROUND(TO_DATE('19-MAR-2017'),'YEAR')
1	01-JAN-17

SELECT TRUNC(TO_DATE('19-MAR-2017'),'MONTH') FROM DUAL;

	TRUNC(TO_DATE('19-MAR-2017'),'MONTH')
1	01-MAR-17

select sysdate+2 from dual;

	SYSDATE+2
1	02-JAN-19

Select sysdate-2 from dual;

	SYSDATE-2
1	29-DEC-18

select to_date('22-dec-18')-to_date('29-DEC-18') from dual;

	TO_DATE('22-DEC-18')-TO_DATE('29-DEC-18')
1	-7

select sysdate-to_date('29-DEC-18') from dual;

	SYSDATE-TO_DATE('29-DEC-18')
1	2.648807870370370370370370370370370370

select sysdate+78/24 from dual;

	SYSDATE+78/24
1	03-JAN-19

select sysdate+2*7 from dual;

	SYSDATE+2*7
1	14-JAN-19

Q5.Write queries to demonstrate all the conversion functions.

select to_char(123)from dual

	TO_CHAR(123)
1	123

Sql:select to_char(SYSDATE,'yyyy')from dual

	TO_CHAR(SYSDATE,'YYYY')
1	2019

select to_char(SYSDATE,'year')from dual

	TO_CHAR(SYSDATE,'YEAR')
1	twenty nineteen

select to_char(SYSDATE,'mm')from dual

	TO_CHAR(SYSDATE,'MM')
1	01

select to_char(SYSDATE,'month')from dual

	TO_CHAR(SYSDATE,'MONTH')
1	january

select to_char(SYSDATE,'mon')from dual

	TO_CHAR(SYSDATE,'MON')
1	jan

select to_char(SYSDATE,'dy')from dual

	TO_CHAR(SYSDATE,'DY')
1	mon

select to_char(SYSDATE,'day')from dual

	TO_CHAR(SYSDATE,'DAY')
1	monday

select to_char(SYSDATE,'dd')from dual

	TO_CHAR(SYSDATE,'DD')
1	21

select to_char(sysdate,'ddspth')from dual

	TO_CHAR(SYSDATE,'DDSPTH')
1	twenty-first

select to_char(sysdate,'ddth')from dual

	TO_CHAR(SYSDATE,'DDTH')
1	21st

select to_char(sysdate,'ddsp')from dual

	TO_CHAR(SYSDATE,'DDSP')
1	twenty-one

select to_char(234,'99999999')from dual

	TO_CHAR(234,'99999999')
1	234

select to_char(234,'0999999')from dual

	TO_CHAR(234,'0999999')
1	0000234

select to_char(234,'$999999')from dual

	TO_CHAR(234,'$999999')
1	$234

select to_char(234,'l999999')from dual

	TO_CHAR(234,'L999999')
1	$234

select to_char(234,'9999.99')from dual

	TO_CHAR(234,'9999.99')
1	234.00

select to_char(13234,'999,999')from dual

	TO_CHAR(13234,'999,999')
1	13,234

select to_number('13234','9999999')from dual

	TO_NUMBER('13234','9999999')
1	13234

Q6.Write queries to demonstrate all the general functions.

select * from cbit

	SID	NAME
1	126	bhargavi
2	(null)	Arya stark

select nvl(sid,129) from cbit

	NVL(SID,129)
1	126
2	129

select nvl2(sid,name,'ting') from cbit

	NVL2(SID,NAME,'TING')
1	bhargavi
2	ting

select NULLIF('jon','jon') from dual

	NULLIF('JON','JON')
1	(null)

select NULLIF('jon','snow') from dual

	NULLIF('JON','SNOW')
1	jon

select coalesce(null,null,'hh') from dual

	COALESCE(NULL,NULL,'HH')
1	hh

AIM: Write the queries for evaluating group functions.

THEORY:

Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group. These functions are: COUNT, MAX, MIN, AVG, SUM, DISTINCT

- COUNT (): This function returns the number of rows in the table that satisfies the condition specified in the WHERE condition. If the WHERE condition is not specified, then the query returns the total number of rows in the table.
- MAX(): This function is used to get the maximum value from a column.
- MIN(): This function is used to get the minimum value from a column.
- AVG(): This function is used to get the averagevalue of a numeric column.
- SUM(): This function is used to get the sum of a numeric column

QUERIES:

Q1.Write queries to demonstrate the count() function.

select count(distinct empname) from emp;

	COUNT(DISTINCTEMPNAME)
1	4

Q2.Write queries to demonstrate the max() function.

select max(empsalary) maxsal from emp;

	MAXSAL
1	63000

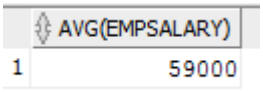
Q3.Write queries to demonstrate the min() function.

select min(empsalary) minsal from emp;

	MINSAL
1	50000

Q4.Write queries to demonstrate the avg() function.

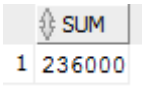
select avg(empsalary) from emp;



	AVG(EMPSALARY)
1	59000

Q5.Write queries to demonstrate the sum() function.

select sum(empsalary) sum from emp;



	SUM
1	236000

AIM: Write the SQL queries using'groupby' function.

THEORY:

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause. We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groupsthe HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

SYNTAX:

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

QUERIES:

Q1.Write a query to display department no, average salary from scott.emp according to department no.

SQL>select deptno,avg(sal) from scott.emp
group by deptno

	DEPTNO	AVG(SAL)
1	30	1566.666666666666666666666666666666666667
2	20	2175
3	10	2916.666666666666666666666666666666666667

Q2.Write a query to display department no, maximum salary, minimum salary from scott.emp according to department no.

SQL>select min(sal),max(sal),deptno from scott.emp
group by deptno;

	MIN(SAL)	MAX(SAL)	DEPTNO
1	950	2850	30
2	800	3000	20
3	1300	5000	10

Q3.Write a query to display department no, maximum salary, minimum salary from scott.emp according to job and department no.

SQL>select min(sal),max(sal),job,deptno from scott.emp
group by job,deptno
order by job;

	MIN(SAL)	MAX(SAL)	JOB	DEPTNO
1	3000	3000	ANALYST	20
2	1300	1300	CLERK	10
3	800	1100	CLERK	20
4	950	950	CLERK	30
5	2450	2450	MANAGER	10
6	2975	2975	MANAGER	20
7	2850	2850	MANAGER	30
8	5000	5000	PRESIDENT	10
9	1250	1600	SALESMAN	30

Q4.Write a query to display department no, having value less than 20 from scott.emp according department no.

SQL>select deptno from scott.emp
where deptno <= 20
group by deptno
order by deptno;

	DEPTNO
1	10
2	20

Q5.Write a query to display department no, average salary where, average salary is more than 2000 from scott.emp department wise.

SQL> select deptno,avg(sal) from scott.emp
group by deptno
having avg(sal)>2000
order by deptno;

	DEPTNO	AVG(SAL)
1	10	2916.66666666666666666666666666666666666667
2	20	2175

Q6.Write a query to display all the department no. from scott.emp where each department is having more than four employees.

SQL>select deptno from scott.emp
group by deptno
having (count(deptno))>=4

	DEPTNO
1	30
2	20

AIM: Write the queries to demonstrate various constraints in SQL.

THEORY:

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Following are some of the most commonly used constraints available in SQL.

- NOT NULL Constraint − Ensures that a column cannot have NULL value.
- UNIQUE Constraint − Ensures that all values in a column are different.
- PRIMARY Key − Uniquely identifies each row/record in a database table.
- FOREIGN Key − Uniquely identifies a row/record in any of the given database table.
- CHECK Constraint − The CHECK constraint ensures that all the values in a column satisfies certain conditions.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

QUERIES:

```
create table student_d
(
sid Number,
sname varchar(10),
sdept varchar(6),
semail varchar(20)
);
insert into student_d
values(&sid,'&sname','&sdept','&semail');
1 row inserted.
alter table student_d
```

Q1.Write queries to demonstrate the primary key constraint.

SQL>

```
add constraint sid_constraint primary key (sid);
Table STUDENT_D altered.

insert into student_d
values(131,'ghut','ihj','jhyuik@gmail.com');
```

Error report -

ORA-00001: unique constraint (CSE_129.SID_CONSTRAINT) violated

Q2.Write queries to demonstrate the UNIQUE constraint.

SQL>

```
alter table student_d
add constraint semail_constraint unique (semail);
```

Table STUDENT_D altered.

```
insert into student_d
values(131,'ghut','ihj','jyo@gmail.com');
```

Error report -

ORA-00001: unique constraint (CSE_129.SID_CONSTRAINT) violated

Q3.Write queries to demonstrate the Foreign key constraint.

SQL>

```
create table id_d
(
iid Number,
year varchar(7)
);
alter table id_d
add constraint iid_constraint primary key(iid);
```

Table ID_D altered.

```
alter table id_d
add constraint fk_constraint foreign key(iid) references student_d(sid);
```

Table ID_D altered.

```
insert into id_d
values(&iid,'&hod');
```

Error report -

ORA-02291: integrity constraint (CSE_129.FK_CONSTRAINT) violated - parent key not found

Q4.Write queries to demonstrate the NOT NULL constraint.

SQL>

```
alter table student_d
```

modify sname varchar(10) NOT NULL;

Table STUDENT_D altered.

insert into student_d

values(&sid,'&sname','&sdept','&semail')

Error report -

ORA-01400: cannot insert NULL into ("CSE_129"."STUDENT_D"."SNAME")

Q5.Write queries to demonstrate the Check constraint.

SQL>

old:insert into student_d

values(&sid,'&same','&sdept','&semail')

new:insert into student_d

values(1433,'fgh','df','erty@hotmail.com')

1 row inserted.

alter table student_d

add constraint check_constraint check (sid<170);

Error report -

ORA-02293: cannot validate (CSE_129.CHECK_CONSTRAINT) - check constraint violated

Q6.Write query to drop the constraint.

SQL>

alter table student_d

drop constraint semail_constraint;

Table STUDENT_D altered.

Q7.Write query to disable the constraint.

SQL>

alter table id_d

disable constraint fk_constraint;

Table ID_D altered.

Q8.Write query to vie the constraints.

SQL>

select constraint_name,constraint_type from user_constraints

	CONSTRAINT_NAME	CONSTRAINT_TYPE
1	SYS_C0013594	V
2	SYS_C0013608	O
3	SID_NN	C
4	SYS_C0011975	P
5	SYS_C0011980	P
6	P_K1	P
7	DPK	P

AIM: Write the queries to demonstrate various joins and views in SQL.

THEORY:

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Here are the different types of the JOINs in SQL:

- CROSS JOIN:Returns all rows in first table joined to all rows in second table.
- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Return all records when there is a match in either left or right table
- EQUI JOIN: Returns records based on the equality condition specified.
- NATURAL JOIN: Automatically joins the matching columns in two tables and retuens matching records.

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

QUERIES:

Q1.Write a query to implement cross product

```
select sid,sname,sdept,year

from student_d s,id_d d;
```

	SID	SNAME	SDEPT	YEAR
1	128	ju	cse	gty
2	129	jyo	ece	gty
3	130	keerth	cse	gty
4	131	kra	it	gty
5	1433	fgh	df	gty
6	128	ju	cse	asd
7	129	jyo	ece	asd
8	130	keerth	cse	asd
9	131	kra	it	asd
10	1433	fgh	df	asd

Q2. Write a query to implement equal join

```
select sid,sname,sdept,year

from student_d s,id_d d

where s.sid=d.iid;
```

SID	SNAME	SDEPT	YEAR
128	ju	cse	gty
129	jyo	ece	asd

Q3. Write a query to implement non equal join

```
select fname,salary,dnumber,dname
from employee1,department1
   where employee1.salary between 10000 and 28000;
```

	FNAME	SALARY	DNUMBER	DNAME
1	Alicia	25000	5	research
2	Alicia	25000	4	administration
3	Alicia	25000	1	headquarters
4	Joyce	25000	5	research
5	Joyce	25000	4	administration
6	Joyce	25000	1	headquarters
7	Ahmad	25000	5	research
8	Ahmad	25000	4	administration
9	Ahmad	25000	1	headquarters

Q4. Write a query to implement right outer join

```
select fname,dnumber,dname
from employee1,department1
   where employee1.DNO=department1.dnumber(+);
```

	FNAME	DNUMBER	DNAME
1	john	5	research
2	Franklin	5	research
3	Alicia	4	administration
4	Jennifer	4	administration
5	Ramesh	5	research
6	Joyce	5	research
7	Ahmad	4	administration
8	James	1	headquarters
9	(null)	3	science

Q5. Write a query to implement cross join

```
select * from section1
cross join course;
```

30 rows selected

Q6. Write a query to implement natural join

select dname,dlocation

from department1 natural join dept_locations;

	DNAME	DLOCATION
1	headquarters	Houston
2	administration	Stafford
3	research	Bellaire
4	research	Sugarland
5	research	Houston

Q7. Write a query to implement self join

select worker.fname|| ' works for '||manager.fname

from employee1 worker,employee1 manager

where worker.sen=manager.super_ssn;

```
WORKER.FNAME||'WORKSFOR'||MANAGER
---------------------------------
Franklin  works for  Joyce
Franklin  works for  Ramesh
Franklin  works for  john
Jennifer  works for  Ahmad
Jennifer  works for  Alicia
James  works for  Jennifer
James  works for  Franklin

7 rows selected.
```

Q8. Write a query to implement join with 'ON' clause

select * from emp join dept

ON(emp.did=dept.did);

eid	ename	mid	did	dname
123	Alekya	126	1	ABC
124	Bhargavi	126	1	ABC
125	Shwetha	126	1	ABC
126	Jyotika	126	1	ABC
127	Vishnu	126	1	ABC
128	Juhitha	131	2	AWC
131	Kranthi	131	2	AWC
142	Susmitha	131	2	AWC
130	Vyshali	140	3	AQY
132	Keerthana	140	3	AQY
133	Keerthi	140	3	AQY
140	Deekshitha	140	3	AQY
141	Ananya	143	4	ADY
143	Ruchitha	143	4	ADY
144	Anmol	143	4	ADY
144	Satwika	143	4	ADY

Q9. Write a query to implement join with 'USING' clause

```
select * from emp join dept
USING(did);
```

eid	ename	mid	did	dname
123	Alekya	126	1	ABC
124	Bhargavi	126	1	ABC
125	Shwetha	126	1	ABC
126	Jyotika	126	1	ABC

127	Vishnu	126	1	ABC
128	Juhitha	131	2	AWC
131	Kranthi	131	2	AWC

AIM: Implementation of PL/SQL queries.

THEORY:

PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL. PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.

- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.

Syntax:

```
Declare
        -variables, cursors, user defined exceptions
Begin
        -SQL statements
        -PL/SQL statements
Exception (optional)
        -Action performed
End;
```

QUERIES:

Q1.Write a query to show the use of case when condition in PL/SQL
SQL>

```
declare
v Number :=88; v1 varchar(1);
v2 varchar(10);

begin
select grade into v1 from sgrade
where (low<=v and high>=v);
dbms_output.put_line(v1);
v2:=case v1
when 'a' then 'Excellent'
when 'b' then 'Good'
when 'c' then 'Average'
end;

dbms_output.put_line(v2);
End;
```

Q2.Write a query to show the use of basic loop in PL/SQL
SQL>

```
declare
i number:=0;

begin
loop
insert into jyo
values(101,'Jyo');
i:=i+1;
exit when i=2;
end loop;

End;
```

Q3.Write a query to show the use of for loop in PL/SQL
SQL>

```
declare

I number:=0;

Begin

for I in 0..4

loop

insert into jyo

values(101,'Jyo');

end loop;


End;
```

Q4.Write a query to print bonus using case when condition in PL/SQL
SQL>

```
set serveroutput on;

Declare

vsal Number;

vBonus Number(10,2);

vid Number:=&vid;


Begin

   select esal into vsal from emp1

   where eid=vid;

   case

   when vsal<5000 then vBonus:=vsal*0.1;

   when vsal<10000 and vsal>5000  then vBonus:=vsal*0.15;

   when vsal>10000 then vBonus:=vsal*0.2;

   else then vBonus:=0;

   end case;

dbms_output.put_line(vBonus);
```

```
End;
```

450

PL/SQL procedure successfully completed.

Q5.Write a query to print '*' for sal-1000 and '' for sal-2000 and so on in PL/SQL**
SQL>

```
set serveroutput on;
Declare
vsal Number;vReward varchar(10);
vid Number:=&vid;

Begin
   select ROUND(esal,-3) into vsal from emp1
   where eid=vid;
   dbms_output.put_line(vsal);
    case vsal
    when 1000 then vReward:='*';
    when 2000 then vReward:='**';
    when 3000 then vReward:='***';
    when 4000 then vReward:='****';
    when 5000 then vReward:='*****';
    when 6000 then vReward:='******';
    when 7000 then vReward:='*******';
    when 8000 then vReward:='********';
    when 9000 then vReward:='*********';
    when 10000 then vReward:='**********';
   else vReward:='';
   end case;
dbms_output.put_line(vReward);
End;
```

(salary is 4500)

5000

*****

PL/SQL procedure successfully completed

Q6.Write a query to copy details of one table into another table in PL/SQL
SQL>

```
declare
vDeptName dept.deptname %type;
counter Number:=1;
vDeptID Number:=1;

Begin
loop
select deptname into vDeptName from dept
where deptid=vDeptID;
insert into deptcopy
values(counter,vDeptName);
counter:=counter+1;
vDeptID:=vDeptID+1;

exit when vDeptID=5;
end loop;
End;
```

PL/SQL procedure successfully completed.

DEPTID	DEPTNAME
1	CSE
2	ECE
3	EEE

4 IT

Q7.Write a query to input the deptno and print the emp details of that deptno using cursors in PL/SQL

SQL>

```
declare
   vDeptno emp.deptno %type:=30;
   vename emp.ename %type;
   veno emp.empno %type;
   i Number :=0;
   j Number :=1;
   cursor cursor_emp is
   select ename,empno from emp
   where deptno=vDeptno;

begin
   open cursor_emp;
   select count(empno) into j from emp
   where deptno=vDeptno;
   loop
   fetch cursor_emp into vename,veno;
   dbms_output.put_line(vename || ' and ' ||veno);
   i:=i+1;
   exit when i=j;
   end loop;
   close cursor_emp;

End;
```

AIM: write the queries for procedures and in pl/sql

THEORY:

PROCEDURES:

A procedure is a program that performs a specific action.A procedure has two parts: specification and body.Procedure specification begins with the keyword 'PROCEDURE' and ends with the procedure 'name' or a 'parameter list'.
The procedure body begins with the keyword 'IS' and ends with the keyword 'END'.
The procedure body has three parts a declarative part, an executable part and an optional exception handling part.

Syntax for creating a procedure:-

```
Procedure name [(parameter list)]
[Local declaration]
Begin
        Executable statements
Exception
        Exception handlers
End [<procedure name>]
```

Calling a procedure:-

```
Procedure_name (parameter list);
```

FUNCTIONS:

A function is a sub program that computes a value.They have a return clause.A function has two parts: specification and body.Function specification begins with the keyword 'FUNCTION' and ends with the 'return clause', which specifies the data type of the return value.The function body begins with the keyword 'IS' and ends with the keyword 'END'.The function body has three parts a declarative part, an executable part and an optional exception handling part.The return statement immediately ends the execution of a sub program and returns control to the caller.Execution continues with the statement following the sub program call.

Syntax for creating a function:-

```
Function <function name> [arguments..]
        Return data type is
[(Local declaration)]
 Begin
        Executable statements
Exception
        Exception handlers
End [<function name>]
```

QUERIES:

Q. A procedure to raise the salary of employee whose employee id is passed to the procedure by Rs 1000/-

Creating a procedure:

```
Create or replace procedure raise_salary (e_id in number)
Is
Sal number (20);
Begin
Update employees set salary=salary+1000 where employee_id=e_id;
If sql%notfound then
raise_application_error (-20101,' with given employee id no data found');
End if;
dbms_output.put_line ('salary updated');
 Select salary into sal from employees where employee_id=e_id;
dbms_output.put_line ('salary after updation is:'||sal);
end;
/
```

Output:
Procedure created.

Using procedure:

```
Declare
Begin
raise_salary (100);
End;
```

Output:
salary updated
salary after updation is:13000
PL/SQL procedure successfully completed.

Q.Write a procedure for formatting given number in (xxx)xxx_xxx.x form:

Creating a procedure:

```
    Create or replace procedure num_format (phno in number)
    Is
        Vphno varchar2 (25);
    Begin
        Vphno: =to_char (phno);
        vphno:='('||substr(vphno,1,3)||')'||substr(vphno,4,3)||'_'||substr(vphno,7,3)||'.'||substr
    (vphno,9,1);
        dbms_output.put_line('After formatting number look like');
        dbms_output.put_line(vphno);
    end;
```

Output:
Procedure created.

Using procedure:

```
declare
begin
        num_format (6573892198);
```

```
end;
```
Output:
After formatting number look like
(657)389_219.8
PL/SQL procedure successfully completed.

Q.Invoking procedure from unknown block & from invoked procedure(P1) invoking other procedure(P2):

Unknown block which invokes procedure p1 ():

```
Declare
Begin
    Dbms_output.put_line ('invoking procedure from unknown block');
    P1 ();
End;
```

Creating a procedure p1 which invokes p2 ():

```
Create or replace procedure P1
Is
Begin
    Dbms_output.put_line ('procedure P1 invoked from unknown block');
    P2 ();
    Dbms_output.put_line ('procedure P1 invoking other procedure P2');
end P1;
```

Creating a procedure p2 ():

```
create or replace procedure P2
is
begin
    Dbms_output.put_line ('procedure P2 invoked by procedure P1');
end;
```

Output:
Invoking procedure from unknown block
Procedure P1 invoked from unknown block
Procedure P2 invoked by procedure P1
Procedure P1 invoking other procedure P2

Statement processed.

Q. Procedure for raising all employees salaries by Rs.1000/-

```
    create or replace procedure raise_salaries
    is
```

```
        cursor cur is select employee_id from employees where employee_id >100 and
    employee_id< 110;
    begin
        for i in cur
        loop
            raise_salary(i.employee_id);
        end loop;
    end;
```

Output:
Procedure created.

Using procedure:

```
set serveroutput on
declare
begin
raise_salaries ();
end;
/
```

Output:

```
salary updated
salary after updation is:19000
salary updated
salary after updation is:19000
salary updated
salary after updation is:11000
salary updated
salary after updation is:8000
salary updated
salary after updation is:6800
salary updated
salary after updation is:6800
salary updated
salary after updation is:6200
salary updated
salary after updation is:14008
salary updated
salary after updation is:11000


PL/SQL procedure successfully completed.
```

Q. Procedure for finding whether the given number is Armstrong or not

Creating a procedure:

```
Create or replace procedure Armstrong (arm in number)
Is
    rem number(5);
    tot number(5):=0;
    num number(5);
BEGIN
    num:=arm;
    while( num > 0 )
    loop
```

```
                rem:=mod(num,10);
                tot:=tot+power(rem,3);
                num:=trunc(num/10);
        End loop;
    if (tot=arm)   then
        dbms_output.put_line(arm||' IS ARMSTRONG NUMBER ');
    else
        dbms_output.put_line(arm||' IS NOT ARMSTRONG NUMBER ');
    End if;
END;
```

Output:

Procedure created.

Using above procedure:

```
declare
begin
Armstrong (153);
end;
```

Output:

153 ARMSTRONG NUMBER

Q. Procedure with 'out' variables(procedure for finding the sum of the digits in a given number

Creating a procedure:

```
create or replace procedure sum_digits(num in number,vout out number)
is
    rem number(5);
    sm number(5):=0;
    num1 number(5);
BEGIN
    Num1:=num;
    while(num1 >0) loop
            rem:=mod(num1,10);
            sm:=sm+rem;
            num1:=trunc(num1/10);
        end loop;
    vout:=sm;
    dbms_output.put_line(sm||' ......'||num||'.........'||vout);
end;
```

Output:

Procedure created.

Using above procedure:

```
Declare
    x number(10);
Begin
    sum_digits (34,x);
    dbms_output.put_line(x);
end;
```

Output:

7 ......34.........7

7

Statement processed.

Queries:

Q. Function which computes and returns some integer value

Creating a function:

```
create or replace function f2(n in number)
return number
is
begin
        return(n*10);
end;
```

output: Function created

Using function:

```
declare
        n number(10);
begin
        n:=f2(100);
        dbms_output.put_line(n);
end;
```

output:

output: 1000
PL/SQL procedure successfully completed.

Q.Calculating tax of all employees(calling one function from another function)

Creating a function which calls another function:

```
Create or replace function sal_tax_of_all_emp
Return number
is
        num number(20):=0;
        cursor cur is select employee_id from employees;
begin
        for i in cur
        loop
                dbms_output.put_line ('tax of employee with id:'||i.employee_id||'  is
        :Rs'||cal_tax (i.employee_id));
        num:=num+1;
        end loop;
```

```
        return (num);
end;
```
creating a function which calculates the tax of the employees
```
create or replace function cal_tax(e_id in number)
return number
is
        sal number(25);
begin
         select salary into sal from employees where employee_id=e_id;
        return(sal*0.08);
end;
```
Using function:
```
declare
begin
        dbms_output.put_line(sal_tax_of_all_emp()||' employees salary taxes is listed');
end;
```

Output:
```
Tax of employee with id :  100  is  :  Rs2160
Tax of employee with id :  101  is  :  Rs1440
Tax of employee with id :  102  is  :  Rs1440
Tax of employee with id :  103  is  :  Rs800
Tax of employee with id :  104  is  :  Rs560
Tax of employee with id :  105  is  :  Rs464
Tax of employee with id :  106  is  :  Rs464
Tax of employee with id :  107  is  :  Rs416
Tax of employee with id :  108  is  :  Rs1040
Tax of employee with id :  109  is  :  Rs800
Tax of employee with id :  110  is  :  Rs736
Tax of employee with id :  111  is  :  Rs696
Tax of employee with id :  112  is  :  Rs704
Tax of employee with id:  113 is:  Rs632
Tax of employee with id :  114 is  :  Rs960
Tax of employee with id :  115  is  :  Rs328
Tax of employee with id :  116  is  :  Rs312
Tax of employee with id :  117  is  :  Rs304
Tax of employee with id :  118  is  :  Rs288
Tax of employee with id :  119  is  :  Rs280
Tax of employee with id :  120  is  :  Rs720
Tax of employee with id :  121  is  :  Rs736
Tax of employee with id :  122  is  :  Rs712
Tax of employee with id :  123  is  :  Rs600
Tax of employee with id :  124  is  :  Rs544
Tax of employee with id :  125  is  :  Rs336
Tax of employee with id :  126  is  :  Rs296
Tax of employee with id :  127  is  :  Rs272
Tax of employee with id :  128  is  :  Rs256
Tax of employee with id :  129  is  :  Rs344
107 employees salary Taxes is listed.
```

Q. Function for checking whether the given number is perfect number

Creating a function:

```
Create or replace function perfct_num (pnum in number)
Return number
is
    sm number(8):=0;
    begin
        for i in 1..pnum-1
      loop
        if mod(pnum,i)=0 then
            sm:=sm + i;
        end if;
    end loop;
    if sm = pnum then
        return 1;
    else
        return 0;
    end if;
end;
```

Output:

Function created….

Calling above function:

```
declare
    bool number;
begin
    bool:=perfct_num(6);
    if bool >= 1 then
        dbms_output.put_line('Given number is perfect number');
    else
        dbms_output.put_line('Given number is not perfect number');
    end if;
end;
```

Output:

Given number is perfect number

Q. Function to display the reverse of a given number.

Creating a function:

```
Create or replace function rev (n in number)
Return number
is
    t number(10):=0;
    r number(10):=0;
    no number(10):=n;
begin
    while no>0
    loop
        t:=mod(no,10);
        r:=r*10+t;
```

```
            no:=no/10;
        end loop;
        return r;
end;
```

Output:

Function created.

Calling a function:

```
declare
        n number(10);
begin
        n:=rev(3404);
        dbms_output.put_line(n);
end;
```

Output:

4043

PL/SQL procedure successfully completed

AIM: write the queries for cursors and in pl/sql

THEORY:

A place where we can have active set of commands.It allows you to name a work area and access its stored information.There are two types of cursors: -

- Implicit cursors
- Explicit cursors

Implicit cursors: -

- Defined by the oracle server.
- They don't have any name.
- It releases the memory allocated when the SQL statements execution is completed.
- In this we can't use open, fetch and close commands.

Explicit cursors: -

- It is defined by the user.
- It is used by the user to process multiple rows returned by a select statement.
- Steps for defining an explicit cursor:
    - Declare the cursor.
    - Open the cursor.
    - Fetch data from the cursor.
    - Close the cursor.

Declaring the cursor:

SynTax: Cursor <cursor name> is SQL statement.

Example: cursor c1 is
Select * from emp where sid between 100 and 110;

- Opening the cursor includes:
  Allocating memory
  Parse the SQL statement for execution.
- The rows which are selected and stored in an Active Set and this active set are stored in the memory. The active set always points to the first row in the active set.
- Fetch: To retrieve data from the active set and is stored in the variables.
  E.g.: fetch ename into a;
- After the fetch instruction the active set is checked. If it is empty it enables the close else it moves the cursor to the next row in the active set and continues until the last row is reached. Once the last row is reaches it enables the close.

- So, close will be enabled in two cases:
    - When active set is empty
    - When active set is pointing to the last row in the active set.

SynTax: close <cursor name>;

Queries:

1.Write a query to show the use of %notfound attribute.

i.Declare
X integer (10);
A char (10);
Name varchar2 (20);
Begin
For I in 140.. 149
Loop
Select first_name into name from employees where employee_id=i;
End loop;
X: =sql%rowcount;
DBMS_OUTPUT.PUT_LINE ('no. of rows selected: '||x);
End;

Output:

No. of rows selected: 1

Statement processed.

ii. declare
vname employees.ename %type;

```
        vsal employees.salary %type;
begin
        select ename,salary into vname,vsal from employees where emp_id=100;
        dbms_output.put_line('No of rows effected    '||to_char(sql%rowcount));
end;
```

Output:

```
No of rows effected 1
PL/SQL procedure successfully completed.
```

2.Explicit cursor [pl/sql block to retrieve 10 employees' details using cursor].

```
Declare
        Vname employees.ename %type;
        Vjid employees.job_id %type;
        Cursor emp_det is
                select ename,job_id into vname,vjid from employees where emp_id between 100
        and 175;
begin
        open emp_det ;
        for i in 1..10
        loop
                fetch emp_det into vname,vjid ;
                dbms_output.put_line (i||'.Name is : '||vname||',Job id is : '||vjid);
        end loop;
        close emp_det;
end;
```

Output:

```
1.Name is : king, Job id is : ad_pres
2.Name is : kochhar, Job id is : ad_vp
3.Name is : de haan, Job id is : ad_vp
4.Name is : hunold, Job id is : it_prog
5.Name is : emst, Job id is : it_prog
6.Name is : mourgos, Job id is : st_man
7.Name is : rajs, Job id is : st_clk
8.Name is : davies, Job id is : st_clk
9.Name is : zlotkey, Job id is : sa_man
10.Name is : able, Job id is : sa_rep
PL/SQL procedure successfully completed.
```

3.Write a pl/sql block using %rowcount attribute in explicit cursor:

```
Declare
        vname employees.ename %type;
        vjid employees.job_id %type;
cursor emp_det is
        select ename,job_id into vname,vjid from employees where emp_id=174 ;
begin
        open emp_det ;
        fetch emp_det into vname,vjid;
        dbms_output.put_line(to_char(emp_det%rowcount));
        close emp_det;
end;
```

Output:

1

PL/SQL procedure successfully completed.

3.Write a pl/sql block using % not found attribute in explicit cursor:

```
declare
        vname employees.ename %type;
        vjid employees.job_id %type;
cursor emp_det is
        select ename,job_id into vname,vjid from employees where emp_id=174 ;
begin
        open emp_det ;
        fetch emp_det into vname,vjid;
        if emp_det%notfound then
                dbms_output.put_line('NOT found is true');
        else
                dbms_output.put_line('NOT found is false');
        end if;
        close emp_det;
end;
```

Output:

NOT found is false

PL/SQL procedure successfully completed.

4.Write a pl/sql block using %found attribute in explicit cursor

```
declare
        vname employees.ename %type;
        vjid employees.job_id %type;
cursor emp_det is
        select ename,job_id into vname,vjid from employees where emp_id=174 ;
begin
        open emp_det ;
        fetch emp_det into vname,vjid;
        if emp_det%found then
                dbms_output.put_line('found is true');
        else
                dbms_output.put_line(' found is false');
        end if;
        close emp_det;
end;
```

Output:

found is true

PL/SQL procedure successfully completed.

5.Write a pl/sql block using parameterized cursor

```
declare
        vname employees.ename %type;
        vjid employees.job_id %type;
cursor emp_det (veid employees. Emp_id %type) is
        select ename,job_id into vname,vjid from employees where emp_id=veid;
begin
```

```
        open emp_det(174) ;
        fetch emp_det into vname,vjid;
        if emp_det%found then
                dbms_output.put_line 'found is true');
                dbms_output.put_line ('1.Name is : '||vname||',Job id is : '||vjid);
        else
                dbms_output.put_line (' found is false');
        end if;
        close emp_det;
end;
```

Output:

found is true

1.Name is : able,Job id is : sa_rep

PL/SQL procedure successfully completed.

ADVANCED CURSOR S:

1.Write a pl/sql block using records in cursor[

```
    declare
        type emp_rec is record(
                e_id number(15),
                d_id number(15),
                d_name varchar2(25));
    cursor cur is
            select emp_id,d.dept_id,d.dept_name from employees e, departments d where
        e.dept_id=d.dept_id;
            vnum number(12):=0;
    begin
        for emp_rec in cur
        loop
                fetch cur into emp_rec;
                vnum:=vnum+1;
        end loop;
    end;
```

Output:

PL/SQL procedure successfully completed.

2.Write a pl/sql block using Cursor with %rowtype (this cursor is to update sailors ratings)

```
    declare
        cursor c_sail is select * from sailors;
        rec_s sailors%rowtype;
    begin
        open c_sail;
        loop
                fetch c_sail into rec_s;
                update sailors set rating=rating+1 where sid=rec_s.sid;
                dbms_output.put_line(rec_s.sid||' '||rec_s.sname||' '|| rec_s.rating);
            exit  when c_sail%notfound;
        end loop;
    end;
```

Output:

31 LUBBER 8

85 ART 3

29 BRUTUS 1
58 RUSSY 10
968 priyank 10
149 ram 10
22 DUSTIN 7
64 HORATIO 7
74 HORATIO 9
95 BOB 3
310 Divya 10
312 priyank 10

3.Write a pl/sql block using cursor with for loop[displaying details of sailors table]

```
    declare
        cursorc_one is (select employee_id,first_name,salary from employees);
begin
        for emp_rec in c_one
        loop
                dbms_output.put_line(emp_rec.employee_id||' '||emp_rec.first_name||'
        '||emp_rec.salary);
        end loop;
end;
```

Output:

100 Steven 24000
101 Neena 17000
102 Lex 17000
103 Alexander 9000
104 Bruce 6000
105 David 4800
106 Valli 4800
107 Diana 4200
108 Nancy 12000
109 Daniel 9000
110 John 8200
123 Shant

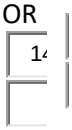
4.Write a pl/sql block using cursor with *for update* clause[update rating of sailors]

```
    declare
        cursor c_sail is select * from sailors
        for update of rating nowait;
                rec_s sailors%rowtype;
    begin
        open c_sail;
        loop
                fetch c_sail into rec_s;
            update sailors set rating=rating+1 where sid=rec_s.sid;
            dbms_output.put_line(rec_s.sid||' '||rec_s.sname||' '|| rec_s.rating);
            exit when c_sail%notfound;
        end loop;
    end;
```

Output:

ORA-00054: resource busy and acquire with NOWAIT specified

OR



31 LUBBER 10
85 ART 9
311 thiru 12
58 RUSSY 12
968 priyank 14
149 ram 12
22 DUSTIN 9
64 HORATIO 9
74 HORATIO 11
95 BOB 5
32 Divya 10
312 priyank 12

5.Write a pl/sql block using cursor with *for update & where current of* clause[updating the rating of sailors in sailors table]

```
declare
        cursor c_sail is select * from sailors
        for update of rating nowait;
begin
        for rec_s in c_sail
        loop
                update sailors set rating=rating+1 where current of c_sail;
                dbms_output.put_line(rec_s.sid||' '||rec_s.sname||' '|| rec_s.rating);
                exit when c_sail%notfound;
        end loop;
end;
```

Output:

31 LUBBER 10
85 ART 9
311 thiru 12
29 BRUTUS 3
58 RUSSY 12
968 priyank 14
149 ram 12
22 DUSTIN 9
64 HORATIO 9
74 HORATIO 11
95 BOB 5
32 ANDY 10
312 priyank 12

AIM: write the queries for triggers and exception handling and in pl/sql

THEORY:

Triggers:

Is a PL/SQL block or a PL/SQL procedure associated with a table, view, schema, or the database. Executes implicitly whenever a particular event takes place

• **Can be either**:
– Application trigger: Fires whenever an event occurs with a particular application
– Database trigger: Fires whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or database

Guidelines for Designing Triggers:

• **Design triggers to**:
– Perform related actions
– Centralize global operations
• **Do not design triggers**:
– Where functionality is already built into the Oracle erver
– That duplicate other triggers

Creating DML Triggers

Triggering statement contains:

• **Trigger timing**

– For table: BEFORE, AFTER
– For view: INSTEAD OF
• **Triggering event**: INSERT, UPDATE, or DELETE
• **Table name**: On table, view
• **Trigger type**: Row or statement
• **WHEN clause**: Restricting condition
• **Trigger body**: PL/SQL blockers

- Before coding the trigger body, decide on the values of the components of the trigger: the trigger timing, the triggering event, and the trigger type.

Exception handling:

- An exception is an identifier which is raised when an oracle error occurs in pl/sql block.
- Warning or error condition is called an exception
- When an error occurs an exception is raised i.e.; the normal execution stops and the control transfers to the exception handling block of the PL/SQL block.
- The 'throw' and 'try' keywords which were normally used in exceptional handling have their functionalities in DBMS as 'raise' and 'exception'.
- Exceptional block is not present in the declare .
- Exceptional block has exceptional handling statements.
- Exceptions can be three types:
    - Predefined exceptions
    - Non predefined exceptions
    - User defined exceptions

Predefined exceptions:

- One of the approximately 20 errors that occurs in oracle PL/SQL block/code
- No need to declare the exception or raise the exception explicitly.
- E.g.: ora_010001 invalid cursor
  Ora_01403 no data found

Non-predefined exception:

- Any other standard errors other than the predefined 20 come under non predefined exceptions.
- Needs to declare the exception explicitly in the declarative section.
- Allow the oracle server to raise that exception implicitly.
- Syntax:
  Exception_name exception;
  PRAGMA_exception_INIT (Exception_name, error number);

User defined exceptions:

- In this, the user has to explicitly declare the exception, raise an exception as well as handle the exception.
- Steps for creating the user defined exception
    - declare the exception
    - raise the exception
    - Use reference name to handle the exception.

Raise_application_error procedure:

- It is used to raise the user defined exceptions.
- It can be used inside PL/SQL block and an exception block

Raise an exception in oracle:

There are two ways to raise an exception.

1. When an oracle error occurs, the associated error is automatically raised by the oracle server.
   E.g. when no roes are selected by the select statement then oracle server raises 'no data found' exception.

2. User can explicitly raise an exception by using 'raise' statement or 'Raise_application_error' statement.

Handling the exception:

1. Trapping the exception: if an exception is raised, control is passed to the exception section of block. The exception will be trapped if it is handled in that exception block itself.

2. Propagating an exception: if the exception is not handled in the outer block, it is propagated / forwarded to the enclosed block.

Queries:

1.write a pl/sql block to fire a trigger upon insertion in employee table

```
create or replace trigger tri_emp

before insert on employee126

begin


if (to_char(sysdate,'dy') in ('sat','sun')) or(to_char(sysdate,'HH24:MI') NOT BETWEEN '08:00'

AND '12:00')

THEN RAISE_APPLICATION_ERROR(-20500,'You may insert into employee table only

during business hours');

end if;

end;
```

Output:

```
Trigger TRI_EMP compiled


Sql->

insert into employee126 values('bavi',7,'23-09-1999','M',23000);
```

Output:

Error starting at line : 8 in command -

insert into employee126 values('bavi',7,'23-09-1999','M',23000)

Error report -

ORA-20500: You may insert into employee table only during business hours

ORA-06512: at 'CSE_126.TRI_EMP', line 3

ORA-04088: error during execution of trigger 'CSE_126.TRI_EMP'

2.write a pl/sql block to fire a trigger uponinsert or update or delete in employee table

```
create or replace trigger tri_emp2
before insert or update or delete on employee126
begin
if (to_char(sysdate,'dy') in ('sat','sun')) or(to_char(sysdate,'HH24:MI') NOT BETWEEN '08:00'
AND '12:00')
THEN
if deleting then
RAISE_APPLICATION_ERROR(-20502,'You may delete from employee table only during
business hours');
elsif inserting then

RAISE_APPLICATION_ERROR(-20503,'You may insert into employee table only during
business hours');
elsif updating('salary') then
RAISE_APPLICATION_ERROR(-20504,'You may update salary only during business hours');
else
RAISE_APPLICATION_ERROR(-20505,'You may update employee table only during business
hours');
end if;
end if;
end;
```

Output:

Trigger TRI_EMP2 compiled

update employee126

set salary=23000 where eid=2;

Output:

Error starting at line : 18 in command -

Error report -

ORA-20504: You may update salary only during business hours

ORA-06512: at 'CSE_126.TRI_EMP2', line 9

ORA-04088: error during execution of trigger 'CSE_126.TRI_EMP2'

delete from employee126

where eid=1;

Output:

Error starting at line : 21 in command -

delete from employee126

where eid=1

Error report -

ORA-20502: You may delete from employee table only during business hours

ORA-06512: at 'CSE_126.TRI_EMP2', line 5

ORA-04088: error during execution of trigger 'CSE_126.TRI_EMP2'

3.write a pl/sql block to fire a trigger upon update of salary in employee table

create or replace trigger restrict_sal

before insert or update of salary on employee126

```
for each row
begin
if not(:new.fname in('Mansa','Fathima'))
AND :NEW.SALARY&gt;15000
THEN
RAISE_APPLICATION_ERROR(-20207,'Employee cannot earn this amount');
end if;
end;
```

Output:

```
Trigger RESTRICT_SAL compiled
```

```
insert into employee126
values('bavi',8,'23-09-1999','F',23000);
```

Output:

```
Error starting at line : 11 in command -
insert into employee126
values('bavi',8,'23-09-1999','F',23000)
Error report -
ORA-20207: Employee cannot earn this amount

ORA-06512: at 'CSE_126.RESTRICT_SAL', line 5
ORA-04088: error during execution of trigger 'CSE_126.RESTRICT_SAL'
```

4.write a pl/sql block to fire a trigger upon insert or update of employee table

And inserting the values into another table using :new and :old keywords

```
create or replace trigger restrict_emp
after delete or insert or update on employee126
for each row
begin insert into nemp values(user,sysdate,:old.eid,:new.eid,:old.fname,:new.fname);
end;
```

Output:

Trigger RESTRICT_SAL compiled

insert into employee126

values('bavi',8,'23-09-1999','F',2000);

Output:

1 row inserted.

select * from nemp;

Output:

	USERNAME	SYSDT	OLDID	OLDNAME	NEWID	NEWNAME
1	CSE_126	18-MAR-19	(null)	8	(null)	bavi

5.write a pl/sql block to fire a trigger upon update of schema

create or replace trigger create_tri

after create on schema

begin

DBMS_OUTPUT.PUT_LINE('Hiii '||user ||' Thank you for creating db on '||sysdate);

end;

Output:

Trigger CREATE_TRI compiled

create table link3

(name varchar2(10));

Output:

Hiii CSE_129 Thank you for creating db on 18-MAR-19

Table LINK3 created.

6.write a pl/sql block to fire a trigger upon logon into schema

```
create or replace trigger logon_tri
after logon on schema
begin
insert into log_trig_table(user_id,log_date,action)
values(user,sysdate,'Logging on');
end;
```

Output:

Trigger LOGON_TRI compiled

After log out and log in:

select * from log_trig_table

Trigger LOGON_TRIG compiled

USER_ID	LOG_DATE	ACTION
CSE_129	18-MAR-19	loggin on

7.write a pl/sql block to fire a trigger upon logon into schema

```
create or replace trigger logoff_tri
after logon on schema
begin
insert into log_trig_table(user_id,log_date,action)
values(user,sysdate,'Logging off');
end;
```

Output:

Trigger LOGOff_TRI compiled

After log out and log in:

select * from log_trig_table;

	USER_ID	LOG_DATE	ACTION
1	CSE_129	18-MAR-19	loggin on
2	CSE_129	18-MAR-19	logging off
3	CSE_129	18-MAR-19	loggin on

Exception handling:

1.write a predefined exception for cursor_already_open:-When you open a cursor that is already open(ORA-06511)

```
Declare
        name employee.ename%type;
        sal employee.salary%type;
cursor c1 is
        select ename, salary from employee
        where eid between 101 and 105;
begin
        open c1;
        loop
                fetch c1 into name,sal;
                Dbms_output.put_line (name||sal);
                exit when c1%notfound;
        end loop;
        open c1;
exception
        when cursor_already_open then
                Dbms_output.put_line ('cursor already open');
end;
```

OUTPUT:

```
Mohan rao 34000
Raja ram   3500
Meena Roy 6500
David     10000
Janaki    20000
Janaki    20000
Cursor already open
 Statement processed.
```

2..write a predefined exception for invalid cursor:-When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened (ORA-01001).

```
Declare
         id number(10);
         cursor c1 is
        select employee_id from employees;
begin
      open c1;
      close c1;
      fetch c1 into id;
 Exception
      when invalid_cursor
      then
            Raise_application_error(-20003,'invalid cursor to fetch');
end;
```

```
 ORA-20003: invalid cursor to fetch
```

3..write a predefined exception for NO_DATA_FOUND:-When a SELECT...INTO clause does not return any row from a table (ORA-01403).

```
set serveroutput on
declare
        vname employ2.lastname%type;
        vsal employ2.salary%type;
begin
        select lastname,salary into vname,vsal from employ2
```

```
        where eid=810;
exception
        when no_data_found then
        DBMS_output.put_line('no rows selected');
end;
no rows selected.
```

4..write a predefined exception for TOO_MANY_ROWS:-When you SELECT or fetch more than one row into a record or variable (ORA-01422).

```
set serveroutput on
declare
        vname employ2.lastname%type;
        vsal employ2.salary%type;
begin
         select lastname,salary into vname,vsal from employ2
        where salary between 10000 and 40000;
exception
         when too_many_rows then
        DBMS_output.put_line('too many rows selected');
end;
too many rows selected
```

5.write a predefined exception for ZERO_DIVIDE:-When you attempt to divide a number by zero (ORA-01476)

```
Declare
        Var1 number (3):=0;
        Var2 number (4):=100;
        vsal employees.salary%type;
Begin
        Update employees
        Set salary= salary+ (var2/var1);
Exception
        When ZERO_DIVIDE then
        Dbms_output.put_line ('divide by zero exception');
end;
Output:
divide by zero exception
Statement processed.
```

6..write a predefined exception for case_not_found:-this exception is raised when the else block isn't use.

```
declare
        name varchar2(10);
        sal number(10,2);
        did number(5);
begin
        select ename,salary,dept_id
        into name,sal,did
        from employee
        where eid=106;
        case
```

```
                when did=204 then dbms_output.put_line('Meeting at 2 p.m.');
                when did=205 then dbms_output.put_line('meeting at 3 p.m.');
        end case;
        DBMS_OUTPUT.PUT_LINE(name||sal||did);
end;
```

OUTPUT:

ORA-06592: CASE not found while executing CASE statement

User defined exceptions:

1.write a userdefined exception for restricting employees based on experience

```
declare
        name varchar2(20);
        experience number(10);
        no_promotion exception;
begin
                select first_name||last_name,round(months_between(sysdate,hire_date)/12,0)
        into name,experience
                from employees
        where employee_id=107;
        case
                when experience>15 then dbms_output.put_line(name||'elligible for promotion
        category');
                when experience=15 then dbms_output.put_line(name||'get prepared for
        promotion test');
                when experience<15 then raise no_promotion;
        end case;
Exception
        when no_promotion
then
                dbms_output.put_line(name||'-experience is='||experience ||'yrs-'||'CANT BE
        PROMOTED---EXPERIENCE < 15yrs');
end;
```

OUTPUT:

DianaLorentz-experience is=13yrs-CANT BE PROMOTED---EXPERIENCE < 15yrs

RAISE_APPLICATION_ERROR:

This is used to convey user define message to the end user.

1.RAISE_APPLICATION_ERROR inside the EXCEPTION BLOCK

```
set serveroutput on
declare
        vname employ2.lastname%type;
        vsal employ2.salary%type;

begin
         select lastname,salary into vname,vsal from employ2
         where eid=810;
exception
        when no_data_found then
        raise_application_error(-20101,'no rows selected');
end;
```

output:

```
declare
*
ERROR at line 1:
ORA-20111: no rows selected
ORA-06512: at line 12
```

2.RAISE_APPLICATION _ERROR using cursors

```
    declare
        vname employ2.lastname%type;
        vsal employ2.salary%type;
        cursor c1 is
                select lastname,salary into vname,vsal from employ2
                 where eid=220;
begin
        open c1;
        fetch c1 into vname,vsal;
         if c1%notfound then
                raise_application_error(-20111,'no rows selected');
        close c1;
        end if;
end;
```

Output:

```
declare
*
ERROR at line 1:
ORA-20111: no rows selected
ORA-06512: at line 12.
```