SEMESTER: **4**

COURSE NO: **CSL 333**

COURSE NAME:

DATABASE LAB

L-T-P: **0-0-4**

HOURS: **4**

CREDIT: **4**

NAME/S OF STAFF:

SHINU MAMACHAN

St. Thomas Institute for Science & Technology
Trivandrum, (College Code: STI)

LAB MANUAL

Department of Computer Science & Engineering

TABLE OF CONTENTS

COURSE PLAN

SL NO	DATE	TITLE	HOURS
1		Design a database schema for an application with ER diagram from a problem description	
2		Creation, modification, configuration, and deletion of databases using SQL Commands	
3		Creation of database schema - DDL (creates tables, set constraints, enforce relationships, create indices, delete and modify tables)	
4		Practice SQL commands for DML (insertion, updating, altering, deletion of data, and viewing/querying records based on condition in databases)	
5		Implementation of built-in functions in RDBMS	
6		Implementation of various aggregate functions in SQL	
7		Implementation of Order By, Group by & Having clause	
8		Implementation of set operators nested queries, and join queries	
9		Practice of SQL TCL commands like Rollback, Commit, and Save point	
10		Practice of SQL DCL commands for granting and revoking user privileges	
11		Practice of SQL commands for creation of views	
12		Implementation of various control structures like IF-THEN, IF-THEN-ELSE, IF-THENELSIF, CASE, WHILE using PL/SQL	
13		Creation of Procedures, Triggers and Functions	
14		Creation of Packages	
15		Creation of Cursors	
16		Creation of PL/SQL blocks for exception handling	

EXPERIMENT NO: 1

ER-DIGRAM

Design an ER diagram for the following requirement:-

A university registrar's office maintains data about the following entities:-

(a) Courses, including numbers, title, credits, syllabus, and prerequisites;

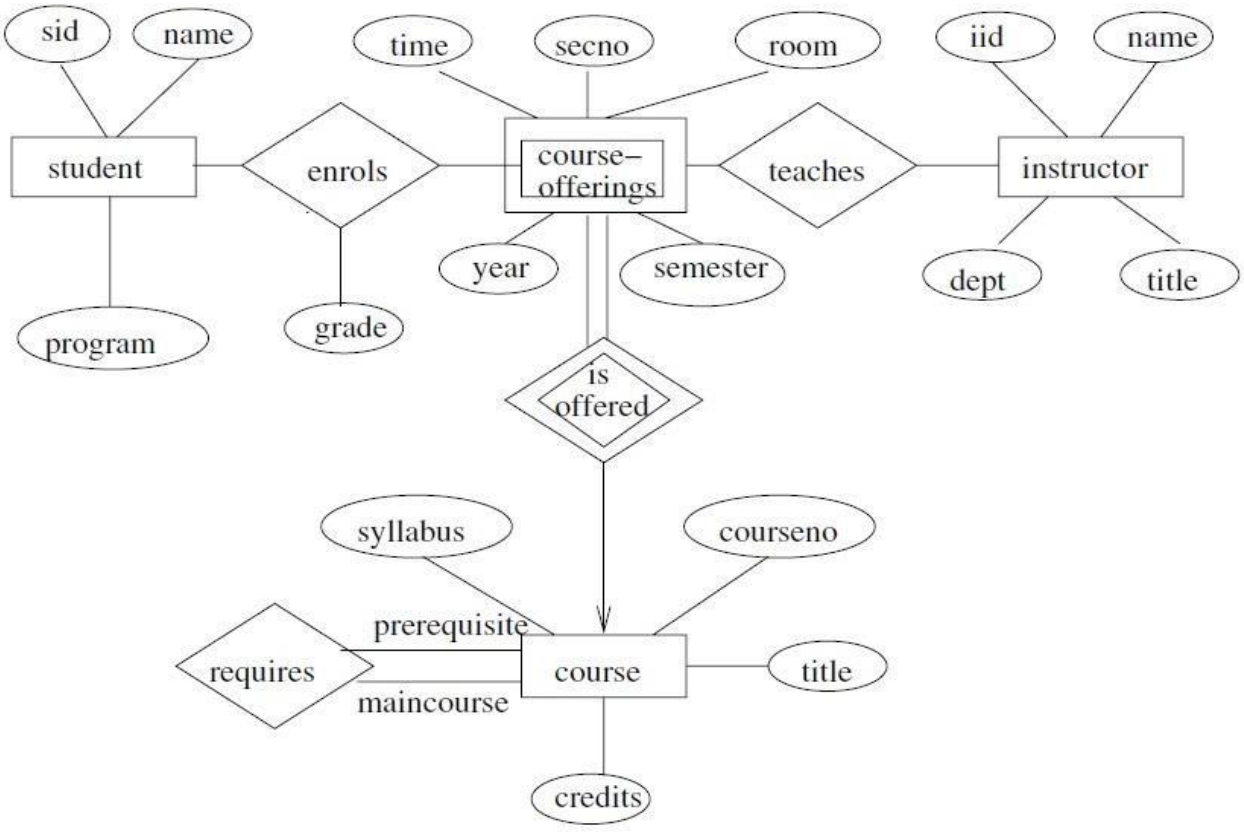
(b) Course offerings, including course number, year, semester, section number, Instructors, timings, and classroom;

(c) Students, including student-id, name, and program;

(d) Instructors, including identification number, name, department, and title.

Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.



EXPERIMENT NO: 2

BASIC SQL COMMANDS

AIM:

To study the basic sql queries such as:

- SELECT
- INSERT
- UPDATE
- DELETE

QUESTIONS

Create a table named Employee and populate the table as shown below.

Emp-id	Emp_name	Dept	Salary (in US $)
1	Michael	Production	2500
2	Joe	Production	2500
3	Smith	Sales	2250
4	David	Marketing	2900
5	Richard	Sales	1600
6	Jessy	Marketing	1800
7	Jane	Sales	2000
8	Janet	Production	3000
9	Neville	Marketing	2750
10	Richardson	Sales	1800

CREATE TABLE Employee (Emp_idint,Emp_namevarchar(15),Deptvarchar(10),Salary int)

```
Table created.
```

INSERT INTO Employee values(1,'Michael','Production',2500)

```
1 row(s) inserted.
```

INSERT INTO Employee values(3,'Smith','Sales',2250)

```
1 row(s) inserted.
```

INSERT INTO Employee values(4,'David','Marketing',2900)

```
1 row(s) inserted.
```

INSERT INTO Employee values(5,'Richard','Sales',1600)

```
1 row(s) inserted.
```

INSERT INTO Employee values(6,'Jessy','Marketing',1800)

```
1 row(s) inserted.
```

INSERT INTO Employee values(7,'Jane','Sales',2000)

```
1 row(s) inserted.
```

INSERT INTO Employee values(8,'Janet','Production',3000)

```
1 row(s) inserted.
```

INSERT INTO Employee values(9,'Neville','Marketing',2750)

```
1 row(s) inserted.
```

INSERT INTO Employee values(10,'Richardson','Sales',1800)

```
1 row(s) inserted.
```

INSERT INTO Employee values(7,'Jane','Sales',2000)

```
1 row(s) inserted.
```

INSERT INTO Employee values(8,'Janet','Production',3000)

```
1 row(s) inserted.
```

INSERT INTO Employee values(9,'Neville','Marketing',2750)

```
1 row(s) inserted.
```

INSERT INTO Employee values(10,'Richardson','Sales',1800)

```
1 row(s) inserted.
```

1. Display the details of all the employees.

SELECT * FROM Employee

EMP ID	EMP NAME	DEPT	SALARY
1	Michael	Production	2500
2	Joe	Production	2500
3	Smith	Sales	2250
4	David	Marketing	2900
5	Richard	Sales	1600
6	Jessy	Marketing	1800
7	Jane	Sales	2000
8	Janet	Production	3000
9	Neville	Marketing	2750
10	Richardson	Sales	1800

2. Display the names and id's of all employees.

SELECT Emp_id,Emp_name FROM Employee

EMP_ID	EMP_NAME
1	Michael
2	Joe
3	Smith
4	David
5	Richard
6	Jessy
7	Jane
8	Janet
9	Neville
10	Richardson

3. Delete the entry corresponding to employee id:10.

DELETE FROM Employee WHERE Emp_id=10

```
1 row(s) deleted.
```

4. Insert a new tuple to the table. The salary field of the new employee should be kept NULL.

INSERT INTO Employee values(10,'Richardson','Sales',NULL)

```
1 row(s) inserted.
```

5. Find the details of all employees working in the marketing department.

SELECT * FROM Employee WHERE Dept='Marketing'

EMP_ID	EMP_NAME	DEPT	SALARY
4	David	Marketing	2900
6	Jessy	Marketing	1800
9	Neville	Marketing	2750

6. Add the salary details of the newly added emloyee.

UPDATE Employee set Salary=1900 WHERE Emp_id=10

```
1 row(s) updated.
```

SELECT * FROM Employee

EMP_ID	EMP_NAME	DEPT	SALARY
1	Michael	Production	2500
2	Joe	Production	2500
3	Smith	Sales	2250
4	David	Marketing	2900
5	Richard	Sales	1600
6	Jessy	Marketing	1800
7	Jane	Sales	2000
8	Janet	Production	3000
9	Neville	Marketing	2750
10	Richardson	Sales	1900

7. Update the salaryof Richard to 1900$.

UPDATE Employee set Salary=1900 WHERE Emp_name='Richardson'

```
1 row(s) updated.
```

SELECT * FROM Employee

EMP_ID	EMP_NAME	DEPT	SALARY
1	Michael	Production	2500
2	Joe	Production	2500
3	Smith	Sales	2250
4	David	Marketing	2900
5	Richard	Sales	1900
6	Jessy	Marketing	1800
7	Jane	Sales	2000
8	Janet	Production	3000
9	Neville	Marketing	2750
10	Richardson	Sales	1900

8. Find the details of all employees who working for marketing and has a salary greater than 2000$.

SELECT * FROM Employee WHERE Dept='Marketing' AND Salary>2000

EMP_ID	EMP_NAME	DEPT	SALARY
4	David	Marketing	2900
9	Neville	Marketing	2750

9. List the names of all employees working in sales department and marketing department.

SELECT emp_name FROM Employee WHERE Dept='Marketing' OR Dept='Sales'

EMP_NAME
Smith
David
Richard
Jessy
Jane
Neville
Richardson

10. List the names and department of all employees whose salary is between2300$ and 3000$.

SELECT Emp_name,Dept FROM Employee WHERE Salary BETWEEN 2300 AND 3000

EMP_NAME	DEPT
Michael	Production
Joe	Production
David	Marketing
Janet	Production
Neville	Marketing

11. Update the salary of all employees working in production department 12%.

UPDATE Employee SET Salary=Salary+salary*0.12 WHERE Dept=’Production’

```
1 row(s) updated.
```

SELECT * FROM Employee

EMP_ID	EMP_NAME	DEPT	SALARY
1	Michael	Production	2800
2	Joe	Production	2800
3	Smith	Sales	2250
4	David	Marketing	2900

5	Richard	Sales	1900
6	Jessy	Marketing	1800
7	Jane	Sales	2000
8	Janet	Production	3360
9	Neville	Marketing	2750
10	Richardson	Sales	1900

12. Display the names of all employees whose salary is less than 2000$ or working for sales department.

SELECT Emp_name FROM Employee WHERE Salary<2000 OR Dept='Sales'

EMP_NAME
Smith
Richard
Jessy
Jane
Richardson

RESULT

The query was executed and the output was obtained.

EXPERIMENT NO: 3

CREATION OF DATABASE SCHEMA USING DDL COMMANDS

AIM:

To solve queries using DDL commands

THEORETICAL BACKGROUND:

SQL (Structured Query Language) is a computer language aimed to store, manipulate, and retrieve data stored in relational databases.

Oracle's database language is **SQL.** In order to communicate with the database, SQL supports the following categories of commands:

- **Data Definition Language (DDL) -** create, alter, truncate and drop commands.
- **Data Manipulation Language (DML) -** insert, select, delete and update commands.
- **Transaction Control Language (TCL) -** Commit savepoint and rollback commands.
- **Data Control Language (DCL) -** grant and revoke commands.

DDL Commands - DDL is used to create an object (eg: table), alter the structure of an object and also to drop the object created.

A **table** is a unit of storage that holds data in the form of rows and columns. DDL is classified as

- create table command
- Alter table command
- Truncate table command
- Drop table command

CREATE Command

Syntax:

create table < table name > (column defn1,column defn2,….);

Eg: create table book (book_id number(5),subject varchar2(25), price number(4,2);

ALTER Command

It is used to modify the structure of the table.

Syntax:

alter table <table name> modify (column defn…);

alter table <table name> add (column defn…);

Eg:

alter table book modify (subject varchar2(30));

alter table book add (pages number(4));

Truncate Command

It is used to delete records stored in a table and the structure has to be retained as it is.

Syntax:

truncate table <table name>;

Eg: truncate table book;

Drop Command

It is used to drop a table.

Syntax:

drop table <table name>

Eg: drop table book;

Constraints

- Integrity constraints are used to enforce business rules on data in the tables.
- Once an integrity constraint is enabled, all data in the table must conform to the rule that it specifies. If you subsequently issue a SQL statement that modifies data in the table, Oracle ensures that the resulting data satisfies the integrity constraint .

Various types of integrity constraints :

- Domain Integrity Constraints.
- Entity Integrity Constraints
- Referential Integrity Constraints.

NOT NULL

NOT NULL is a constraint which can be used to avoid the entry of null values into the rows. This means, when you construct a not null constraint while creating table, this will not allow you to leave the rows without entering data.

Example :

CREATE TABLE customer (custno number(3) not null, name varchar2(20) not null, phone number(7), sex varchar2(8) not null);

CHECK Constraints

- This Check constraint is used to assign a constraint to prevent the violation of entries.
- This is a condition to check whether the entry value is matching to the constraint or not. If it is not matching this will not allow you to enter against the constraint.

Syntax

CREATE TABLE <table name> (column name <data type> (size) constraint < constraint name < condition >)

Example

CREATE TABLE temp (stud_id number(3), name varchar2(10), sal number(10,2) constraint chk_sal check (sal >1000));

To assign the Check constraint to the existing table :

Syntax :

Alter table <table_ name> add constraint < constraint name> < condition >)

Example :

Alter table customer add constraint chksal check (sal >1000);

Alter table order_master add constraint chk1 check(o_status in ('p','c'));

To Disable the Check Constraints :

Syntax;

ALTER TABLE <table name > disable constraint <constraint name>

Example;

ALTER TABLE Customer DISABLE constraint Chk_sal;

Entity integrity constraints

Following Constraints are coming under entity integrity constraints.

1. PRIMARY KEY
2. UNIQUE

Primary Key:

- A Primary Key allows each row in a table to be uniquely identified and ensures that no duplicate rows exist and no null values are entered.
- A Table can have one primary key.
- A Primary key value is only used to identify a row in the table

Syntax;

Create table <table name> <column name (data type (size)) constraint <constraint name> primary key);

Example;

Create table customer (custno number(3) constraint PK_CUST primary key, cname varchar2(20) not null, Age number(2) not null);

or

Create table customer (custno number(3), cname varchar2(20) not null, Age number(2) not null, constraint PK_CUST primary key(custno));

or

Create table customer (custno number(3) primary key, cname varchar2(20));

UNIQUE:

- Unique forces the column values to be unique.
- More than one unique constraint can be given on a table
- Unique constraints will allow nulls while preventing duplicate values column.

Example;

Create table customer (custno number(3) unique , cname varchar2(20));

Referential Integrity Constraint

Foreign Key :

The two tables are related by a common column (or set of columns), define a PRIMARY or UNIQUE key constraint on the column in the parent table, and define a FOREIGN KEY constraint on the column in the child table, to maintain the relationship between the two table.

FOREIGN KEYS can be comprised of multiple columns. However, a composite foreign key must reference a composite PRIMARY or UNIQUE KEY of the exact same structure that is the same number of columns and data types. Because Composite primary Key and Unique keys are limited to 16 columns, a composite foreign key is also limited to 16 columns.

Example;

To ensure that each employee in the EMP table works for a department that is listed in the DEPT table, first create a Primary Key constraint on the DEPTNO column of the DETP table with this statement.

ALTER TABLE dept add PRIMARY KEY (deptno);

Then create the referential integrity constraint on the DEPTNO column of the EMP table that references the Primary Key of the DEPT table:

ALTER TABLE emp add FOREIGN KEY (deptno) REFERENCES dept(deptno);

If you subsequently add a new employee record to the table, oracle automatically ensures that its department number appears in the department table.

Questions:

1). Create a table with name customer and fields as follows:

Cust_id number

cust_name varchar2(20)

cust_hname varchar2(20)

cust_street varchar2(20)

cust_phone integer.

create table customer(cust_id integer,cust_name varchar(20),cust_hname varchar2(20),cust_street varchar2(20),cust_phone integer);

2) Create table order with fields as follows

Orderid number

Custid number

Itemcode number

Orderquantity number

Deliverydate date

Modeof payment char(1)

create table order_db(order_id number, cust_id number, item_code number(5), order_date date, ex

piry_date date, delivery_date date, payment_mode char(1));

3) Create table item_db with the following fields

Item_code number

Item_name char(1)

Current_stock number

Unitprice number

create table item_db(item_code number(5), item_name varchar2(20), stock number(5), unit_price n

umber(5));

4). Alter the table to add a primary key to customer_id in customer table.

alter table customer add constraint p1k primary key(cus_id);

5). Alter the table to add a primary key to itemcode in item_details.

alter table itemtable add constraint p2k primary key(itemcode);

6). Alter the table to set foreign key constraint in order_db to link with customer_db using customer_id field.

alter table order_db add constraint foreign key references customer_db(cust_id);

7). Add a not null constraint to the field date_of_order.

alter table order_db modify order_date constraint c1 not null;

8). Add a unique constraint for the order_id in order_db.

alter table order_db modify order_id constraint c2 unique;

9). Add a check constraint to the field mode_of _payment in order_db such that it takes three values'D' (for cheque), 'R' (for cash) and 'C' (for credit card).

alter table order_db add constraint c3 check (payment_mode in ('D', 'R', 'C'));

10). Add a new field 'remark' as char(10) to the order_db and set the default value , as 'direct'.

alter table order_db add remark char(10) default 'direct';

11). Modify the data type and size of remark as varchar2 (15) in order_db.

alter table order_db modify remark varchar2(10);

12). Delete the column ‘remark’ from order_db.

alter table order_db drop column remark;

13). Describe all tables and delete the table ‘test_db’.

desc customer_db;

Name	Null?	Type
CUST_ID	NOT NULL	NUMBER(5)
CUST_NAME		VARCHAR2(20)
HOUSENAME		VARCHAR2(20)
STREET		VARCHAR2(20)
PHONENO		NUMBER

create table test_db(t1 varchar2(10), t2 number, t3 date, t4 char(5));

---

drop table test_db;

RESULT:

The query was executed successfully and output was verified.

EXPERIMENT NO: 4

DML COMMANDS

AIM:

To solve queries using DML Commands

THEORETICAL BACKGROUND:

Data Manipulation Language

Data manipulation languages are used to query and manipulate existing objects like tables. The DML commands are

1. Insert
2. Select
3. Update
4. Delete

INSERT COMMAND

INSERT command is used to add one or more rows to a table. While using this command the values are separated by commas and the data types varchar2, char, long and date are enclosed in single quotes. The values must be entered in the same order as they are defined in the table.

Syntax:

insert into <table name> values (a list of data values);

Eg :

insert into book values (123 ,'Database',45.33);

insert into book (book_id, subject, price) values (124,'C Language',500);

insert into book values (&book_id, '&subject', &price);

insert into book values(120,'Compiler',to_date('2009/01/09:02:20:05PM', 'yyyy/mm/dd:hh:mi:sspm'));

SELECT COMMAND

SELECT command is used for requesting information stored in a table .

Syntax:

Select <column _name> from <table name>;

Eg: select book _id, price from book;

a. Selecting distinct rows

To prevent the selection of duplicate rows, we include **distinct** clause in the select command.

Syntax:

Select distinct <column name> from <table name>;

Eg: Select distinct subject from book;

b. Select with where clause

To select specific rows from a table we include a 'where' clause in the select command.

To arrange the displayed rows according to some pre-defined order we use 'order by' clause.

Syntax:

Select <columns> from <table name> where <conditions> [order by <column>];

Eg: select * from book where price = 30.00 order by book _id;

UPDATE COMMAND

Update command is used to change the existing records in a table. The update command consists of a 'set' clause and an optional 'where' clause.

Syntax:

update <table name> set field = value [where <condition>];

Eg: update book set book_id = 45;

DELETE COMMAND

Delete command is used for deleting rows.

Syntax:

Delete from <table name> [where <conditions>];

Eg: delete from book where book_id = 40;

Questions:

1. Insert a single record in the following tables
   Customer_db, item_db, order_db

insert into customer_db values(1000, 'Ajith', 'Karthika', 'N C C Road', 2437189);

insert into order_db values(500, 1000, 143, '9-mar-09', '31-dec-99', '9-mar-09', 'R');

insert into item_db values(143, 'RSS', 1, 10000);

2. Insert multiple records in the following tables
   Customer_db, item_db, order_db

insert into customer_db values(&cust_id, '&cust_name', '&housename', '&street', &phoneno);

Enter value for cust_id: 1006

Enter value for cust_name: Devi

Enter value for housename: Karthika

Enter value for street: N C C Road

Enter value for phoneno: 2438959

old 1: insert into customer_db values(&cust_id, '&cust_name', '&housename', '&street', &phoneno)

new 1: insert into customer_db values(1006, 'Devi', 'Karthika', 'N C C Road', 2438959)

3. update the delivery date in the field order_db as delivery_date= date of order +10;

update order_db set delivery_date=order_date+10;

4. Display the records in ascending order by delivery date in order_db

select * from order_db order by delivery_date asc;

ORDER_ID	CUST_ID	ITEM_CODE	ORDER_DAT	EXPIRY_DA	DELIVERY_	P
333	1001	10	05-AUG-09	31-DEC-10	15-AUG-09	D
101	1001	12	05-AUG-09	31-DEC-10	15-AUG-09	D

5. Display the records in descending order by unit price in item_db

select * from item_db order by unit_price desc;

ITEM_CODE	ITEM_NAME	STOCK	UNIT_PRICE
143	RSS	1	10000
12	Pen	50	20
10	eyChain	100	15

6.Display the records in order_db where order_id >1 and <7

select * from order_db where order_id between 1 and 7;

ORDER_ID	CUST_ID	ITEM_CODE	ORDER_DAT	EXPIRY_DA	DELIVERY_	P
3	1002	6	01-JAN-09	02-JAN-09	11-JAN-09	C
2	1003	20	02-JAN-09	30-MAR-09	12-JAN-09	C

7. Delete the order from order_db where expiry date < current date

```
delete from order_db where expiry_date<'11-aug-09';
```

8.Duplicate the table of item_db as test_db

```
create table test_db as select * from item_db;
```

9. Select customer name from customer_db whose anme starts with alphabet A. select cust_name from customer_db where cust_name like 'A%';

```
CUST_NAME
---------------
Ajith
Akhilesh
Alfred
Alveena
Aneese
```

10. Display order_id and cust_id from order_db

```
select order_id, cust_id from order_db;
```

```
  ORDER_ID    CUST_ID
---------- ----------
       333       1001
       101       1001
```

11. Truncate the table item_db

```
    truncate table item_db;
```

12. Display order_id , customer_id from order_id whose month of delivery is current month.

select order_id, cust_id from order_db where delivery_date like '%-AUG-%';

```
  ORDER_ID    CUST_ID
---------- ----------
       333       1001
       101       1001
```

13 .Display orderid and customerid from order database

SQL> select orderid,custid from order1;

```
   ORDERID     CUSTID
---------- ----------
       101       4677
       128       4299
```

14. Truncate the table

SQL> truncate table item_db;

Table truncated.

SQL> select * from item_db;

no rows selected

15.Display the orderid,customerid from orders where the month of delivery is the current month.

SQL> update order1 set deliverydate='01-JUL-2015' where deliverydate='17-jan-15';

1 row updated.

SQL> select * from order1;

ORDERID CUSTID ITEMCODE ORDERQUANTITY DELIVERYD M EXPIRYDAT

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

101 4677 1001 6 01-JUL-15 C 26-JAN-15

128 4299 1002 16 23-APR-15 D 31-DEC-15

SQL> select * from order1 where extract(month from deliverydate)=to_char(SYSDATE,'MM');

ORDERID CUSTID ITEMCODE ORDERQUANTITY DELIVERYD M EXPIRYDAT

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

101 4677 1001 6 01-JUL-15 C 26-JAN-15

16. Find the average of order quantity with item code =256.

SQL> insert into order1 values(150,4677,256,102,'29-may-2015','R','2-jul-2015');

1 row created.

SQL>insert into order1 values(151,4299,256,6,'31-may-2015','C','2-jul-2015');

1 row created.

SQL> select * from order1;

ORDERID CUSTID ITEMCODE ORDERQUANTITY DELIVERYD M EXPIRYDAT

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

101 4677 1001 6 01-JUL-15 C 26-JAN-15

128 4299 1002 16 23-APR-15 D 31-DEC-15

150 4677 256 102 29-MAY-15 R 02-JUL-15

151 4299 256 6 31-MAY-15 C 02-JUL-15

SQL> select avg(orderquantity) from order1 where itemcode=256;

AVG(ORDERQUANTITY)

54

17. What is the item with the highest unit price?

SQL> select * from item_db where unitprice=(select max(unitprice) from item_db);

ITEM_CODE I CURRENT_STOCK UNITPRICE

--- - --- ---

150 Y 25 30

18. What is the cheapest item?

SQL> select * from item_db where unitprice=(select min(unitprice) from item_db);

ITEM_CODE I CURRENT_STOCK UNITPRICE

--- - --- ---

256 X 150 22

19. How many orders were made for item with itemcode 250

SQL> select count(*) from order1 where itemcode=256

COUNT(*)

2

20. How many items have unit price between 100 and 200.

SQL> select count(*) from item_db where unitprice>100 and unitprice<200;

```
  COUNT(*)
----------
         0
```

21. What is the average unit price?

SQL> select avg(unitprice) from item_db;

```
AVG(UNITPRICE)
--------------
            26
```

22. Display the orderid and delivery date with heading prdercode and date of delivery.

SQL> select orderid as order_code,deliverydate as dateofdelivery from order1;

```
ORDER_CODE DATEOFDEL
---------- ---------
       101 01-JUL-15
       128 23-APR-15
       150 29-MAY-15
       151 31-MAY-15
```

23. Display the name of customers which contain occurances of a and j in the same name.

SQL> insert into customer values(4618,'arjun','abc','xyz',9447455934)

1 row created.

SQL> insert into customer values(4186,'Jasmine','abc','xyz',123456789);

1 row created.

SQL> select cust_name from customer where cust_name like '%a%j%' or cust_name like '%J%a%'

CUST_NAME

---

arjun

Jasmine

24 .What is the length of shortest name?

SQL> select(length(min(CUST_NAME))) from customer;

(LENGTH(MIN(CUST_NAME)))

4

25. Create table deliverdb with same structure as orderdb.

SQL> create table deliver_db as select * from order1;

Table created.

SQL> select * from deliver_db;

ORDERID CUSTID ITEMCODE ORDERQUANTITY DELIVERYD M EXPIRYDAT

---

101 4677 1001 6 01-JUL-15 C 26-JAN-15

128 4299 1002 16 23-APR-15 D 31-DEC-15

150 4677 256 102 29-MAY-15 R 02-JUL-15

151 4299 256 6 31-MAY-15 C 02-JUL-15

26. Display the records of tables orderdb deliverydb using union operator

SQL> insert into deliver_db values(176,4911,1001,5,'05-JUL-15','C','29-JUL-15');

1 row created

SQL> select * from deliver_db;

ORDERID CUSTID ITEMCODE ORDERQUANTITY DELIVERYD M EXPIRYDAT

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

101 4677 1001 6 01-JUL-15 C 26-JAN-15

128 4299 1002 16 23-APR-15 D 31-DEC-15

150 4677 256 102 29-MAY-15 R 02-JUL-15

151 4299 256 6 31-MAY-15 C 02-JUL-15

176 4911 1001 5 05-JUL-15 C 29-JUL-15

SQL> select * from deliver_db union select * from order1;

ORDERID CUSTID ITEMCODE ORDERQUANTITY DELIVERYD M EXPIRYDAT

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

101 4677 1001 6 01-JUL-15 C 26-JAN-15

128 4299 1002 16 23-APR-15 D 31-DEC-15

150 4677 256 102 29-MAY-15 R 02-JUL-15

151 4299 256 6 31-MAY-15 C 02-JUL-15

176 4911 1001 5 05-JUL-15 C 29-JUL-15

27. Display the records having order id common for both tables orderdb and deliverydb using intersect operator

SQL> select orderid from deliver_db intersect select orderid from order1;

ORDERID

101

128

150

151

28. Display the orderid of the order that is not delivered yet.

SQL> insert into deliver_db values(192,4299,1001,2,'08-JUL-15','C','12-JUL-15');

1 row created.

SQL> select * from deliver_db;

ORDERID CUSTID ITEMCODE ORDERQUANTITY DELIVERYD M EXPIRYDAT

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

101 4677 1001 6 01-JUL-15 C 26-JAN-15

128 4299 1002 16 23-APR-15 D 31-DEC-15

150 4677 256 102 29-MAY-15 R 02-JUL-15

151 4299 256 6 31-MAY-15 C 02-JUL-15

176 4911 1001 5 05-JUL-15 C 29-JUL-15

192 4299 1001 2 08-JUL-15 C 12-JUL-15

SQL> select orderid from deliver_db where deliverydate>SYSDATE;

ORDERID

192

RESULT:

The query was executed successfully and output was verified.

EXPERIMENT NO: 5

IMPLEMENTATION OF BUILT IN FUNCTIONS IN RDBMS

AIM:

To understand and implement various types of function in SQL

- Number Function
- Character Function
- Conversion Function
- String Functions
- Date

[Note : Dual is Built in default database]

NUMBER FUNCTION

1. **Abs(n)** : Select abs(-15) from dual;
2. **Exp(n)** : Select exp(4) from dual;
3. **Power(m,n)** : Select power(4,2) from dual;
4. **Mod(m,n)** : Select mod(10,3) from dual;
5. **Round(m,n)** : Select round(100.256,2) from dual;
6. **Trunc(m,n)** : Select trunc(100.256,2) from dual;
7. **Sqrt(m,n)** : Select sqrt(16) from dual;.

CHARACTER FUNCTION

8. **initcap(char)** **: select initcap(“hello”) from dual;**
9. **lower (char)** **: select lower (‘HELLO’) from dual;**

10. **upper (char)** **:select upper (‘hello’) from dual;**
11. **ltrim (char,[set])** **: select ltrim (‘cseit’, ‘cse’) from dual;**
12. **rtrim (char,[set])** **: select rtrim (‘cseit’, ‘it’) from dual;**
13. **replace (char,search string,replace string)** **: select replace(‘jack and jue’,‘j’,‘bl’) from dual;**

CONVERSION FUNCTIONS:

14. **To_char:** TO_CHAR (number) converts n to a value of VARCHAR2 data type, using the optional number format fmt. The value n can be of type NUMBER, BINARY_FLOAT, orBINARY_DOUBLE.

    SQL>select to_char(65,'RN')from dual;

    LXV

15. **To_number** : TO_NUMBER converts expr to a value of NUMBER data type.

    SQL>Select to_number ('1234.64') from Dual;

    1234.64

16. **To_date**:TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or

    NVARCHAR2 data type to a value of DATE data type.

    SQL>SELECT TO_DATE('January 15, 1989, 11:00 A.M.')FROM DUAL;

    TO_DATE

    15-JAN-89

STRING FUNCTIONS

1. **Concat**: CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes

   SQL>SELECT CONCAT('ORACLE','CORPORATION')FROM DUAL;

   ORACLECORPORATION

18. **LPAD** returns expr1, left-padded to length n characters with the sequence of characters in expr2.

    SQL>SELECT LPAD('ORACLE',15,'*')FROM DUAL;

    *********ORACLE

19. **RPAD** returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary.

    SQL>SELECT RPAD ('ORACLE',15,'*')FROM DUAL;

    ORACLE*********

20. **Ltrim**: Returns a character expression after removing leading blanks.

    SQL>SELECT LTRIM('SSMITHSS','S')FROM DUAL;

    MITHSS

21. **Rtrim**: Returns a character string after truncating all trailing blanks

    SQL>SELECT RTRIM('SSMITHSS','S')FROM DUAL;

    SSMITH

22. **Lower**: Returns a character expression after converting uppercase character data to lowercase.

    SQL>SELECT LOWER(‘DBMS’)FROM DUAL;

    Dbms

23. **Upper**: Returns a character expression with lowercase character data converted to uppercase

    SQL>SELECT UPPER(‘dbms’)FROM DUAL;

    DBMS

24. **Length:** Returns the number of characters, rather than the number of bytes, of the given

    string expression, excluding trailing blanks.

    SQL>SELECT LENGTH(‘DATABASE’)FROM DUAL;

25. **Substr:** Returns part of a character, binary, text, or image expression.

    SQL>SELECT SUBSTR(‘ABCDEFGHIJ’3,4)FROM DUAL;

    CDEF

26. **Instr:** The INSTR functions search string for substring. The function returns an integer

    indicating the position of the character in string that is the first character of this occurrence.

SQL>SELECT INSTR('CORPORATE FLOOR','OR',3,2)F

ROM DUAL;

14

DATE FUNCTIONS:

27. **Sysdate:**

SQL>SELECT SYSDATE FROM DUAL;

29-DEC-08

28. **next_day:**

SQL>SELECT NEXT_DAY (SYSDATE,'WED') FROM DUAL;

05-JAN-09

29. **add_months:**

SQL>SELECT ADD_MONTHS(SYSDATE,2)FROM DUAL;

28-FEB-09

30. **last_day:**

SQL>SELECT LAST_DAY(SYSDATE)FROM DUAL;

31-DEC-08

31. **months_between**:

SQL>SELECT MONTHS_BETWEEN (SYSDATE,HIRE DATE)FROM EMP;

4

32. **Least**:

SQL>SELECT LEAST('10-JAN-07','12-OCT-07')FROM DUAL;

10-JAN-07

33. **Greatest:**

SQL>SELECT GREATEST('10-JAN-07','12-OCT-07')FROM DUAL;

10-JAN-07

34. **Trunc:**

SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;

28-DEC-08

35. **Round**:

SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL;

28-DEC-08

36. **to_char:**

SQL> select to_char(sysdate, "dd\mm\yy") from dual;

24-mar-05.

37. **to_date**:

SQL> select to date (sysdate, "dd\mm\yy") from dual;

24-mar-o5.

Create a table named *acct_details* and populate the table as shown below.

ATM_No	Account_Holder	Valid_From	Expiry_Date
4375-0107-2415-1578	JAMES MATHEW	05-08-10	05-08-22
3241-1217-2416-1670	THOMAS VARGHESE	02-09-12	02-09-24
4352-2327-4717-1773	TIJO SAMUEL	21-12-07	21-12-19
5532-3435-5316-1876	ALICE THOMAS	17-01-13	17-01-25
3652-4543-6414-1975	NEVIN KOSHY	19-02-09	19-02-21
8324-5657-7018-1073	NIRMAL GEORGE	16-05-11	16-05-23
4205-6760-4412-1176	SAMPATH SIMON	11-08-08	11-08-20
0083-7873-0214-1274	BIBIN TOM	29-11-06	29-11-18
3347-8987-2410-1372	BINCY SAMUEL	20-10-05	20-10-17
1629-9107-4412-1472	GRACE ALEX	08-08-08	08-08-20

1. Find out how many months have passed since Alice has started an account
2. Find the account holder who has started the account most recently
3. Find out how many months have passed since Alice has started an account
4. How many days are remaining for Nirmal's ATM card to get expired
5. The Bank has decided to extend the expiry dates for a few of its early customers.All customers who have started their account before January 1 2009 gets their expiry date extended by 1 year. Update the database.
6. Display the current system date and show the remaining days for expiry of each account holder.
7. Whose ATM card has got the greatest deadline for expiry?
8. The Valid-From date of Grace Alex was wrongly entered by the bank. It should be replaced with the next day.
9. Display the details corresponding to all customers whose account were started in the current year
10. Find the age of Bincy Samuel if her account was started when she was 18
11. Using extract function display expiry year for all the acct_hldrs.

RESULT:

The query was executed successfully and output was verified.

EXPERIMENT NO: 6

AGGREGATE FUNCTION IN SQL

AIM:

Introduction to Aggregate functions
-AVG() -MAX() -MIN() - COUNT() -SUM()\

Description:

Aggregate Functions;

Sum()
Avg()
Count()
Max()
Min()

- **Sum(*fieldname*)**

  Returns the total sum of the field.

- **Avg(*fieldname*)**

  Returns the average of the field.

- Count( )

  Count function has three variations:

  Count(*) : returns the number of rows in the table including duplicates and those with null values

  Count(*fieldname*) : returns the number of rows where field value is not null Count (All): returns the total number of rows.

  It is same like count(*)

- **Max(*fieldname*)**

  Returns the maximum value of the field

- **Min(*fieldname*)**

  Returns the maximum value of the field

QUESTION

Create a table named student and populate the table as shown in the table.

The table contains the marks of 10 students for 3 subjects(Physics, Chemistry, Mathematics).The total marks for physics and chemistry is 25.while for mathematics it is 50.The pass mark for physics and chemistry is 12 and for mathematics it is 25.A student is awarded a 'Pass' if he has passed all the subjects.

Roll No	Name	Physics	Chemistry	Maths
1	Adam	20	20	33
2	Bob	18	9	41
3	Bright	22	7	31
4	Duke	13	21	20
5	Elvin	14	22	23
6	Fletcher	2	10	48
7	Georgina	22	12	22
8	Mary	24	14	31
9	Tom	19	15	24
10	Zack	8	20	36

CREATE TABLE student(roll_noint,namevarchar(10),physics int,chemistryint,mathsint)

```
Table created.
```

INSERT INTO student values(1,'Adam',20,20,33)

```
1 row(s) inserted.
```

INSERT INTO student values(2,'Bob',18,9,41)

```
1 row(s) inserted.
```

INSERT INTO student values(3,'Bright',22,7,31)

```
1 row(s) inserted.
```

INSERT INTO student values(4,'Duke',13,21,20)

```
1 row(s) inserted.
```

INSERT INTO student values(5,'Elvin',14,22,23)

```
1 row(s) inserted.
```

INSERT INTO student values(6,'Fletcher',2,10,48)

```
1 row(s) inserted.
```

INSERT INTO student values(7,'Georgina',22,12,22)

```
1 row(s) inserted.
```

INSERT INTO student values(8,'Mary',24,14,31)

```
1 row(s) inserted.
```

INSERT INTO student values(9,'Tom',19,15,24)

```
1 row(s) inserted.
```

INSERT INTO student values(10,'Zack',8,20,36)

```
1 row(s) inserted.
```

SELECT * FROM student

ROLL_NO	NAME	PHYSICS	CHEMISTRY	MATHS
1	Adam	20	20	33
2	Bob	18	9	41
3	Bright	22	7	31
4	Duke	13	21	20
5	Elvin	14	22	23
6	Fletcher	2	10	48
7	Georgina	22	12	22
8	Mary	24	14	31
9	Tom	19	15	24
10	Zack	8	20	36

1.Find the class average for the subject 'Physics'

SELECT avg(Physics)from student

AVG(PHYSICS)
15.125

2.Find the highest marks for mathematics (To be displayed as highest_marks_maths).

SELECT max(maths)as Highest_marks_maths from student

HIGHEST_MARKS_MATHS
48

3.Find the lowest marks for chemistry(To be dispayed as lowest_mark_chemistry)

SELECT min(chemistry)as lowest_mark_chemisry from student

LOWEST_MARK_CHEMISRY
7

4.Find the total number of students who has got a 'pass' in physics.

SELECT COUNT(physics)from student WHERE physics>=12

COUNT_ROLLNO
8

5.Generate the list of students who have passed in all the subjects

SELECT * FROM student where physics>=12 AND chemistry>=12 AND maths>=25

ROLL_NO	NAME	PHYSICS	CHEMISTRY	MATHS
1	Adam	20	20	33
8	Mary	24	14	31

6.Generate a ranklist for the class.Indicate Pass/Fail. Ranking based on total marks obtained by the students.

ALTER table studentADD totalmarkintADD RESULT varchar(10)

```
Table altered.
```

UPDATE student set totalmark =physics+chemistry+maths

```
Table updated.
```

UPDATE student set RESULT='P' WHERE physics>=12 AND chemistry>=12 AND maths>=25

```
Table updated.
```

UPDATE student set RESULT='F' WHERE physics<12 OR chemistry<12 OR maths<25

```
Table updated.
```

SELECT * from student order by total desc.

ROLL_N O	NAME	PHYSIC S	CHEMISTR Y	MATH S	TOTALMAR K	RESUL T
1	Adam	20	20	33	73	P
8	Mary	24	14	31	69	P
2	Bob	18	9	41	68	F
10	Zack	8	20	36	64	F
3	Bright	22	7	31	60	F
6	Fletcher	2	10	48	60	F
5	Elvin	14	22	23	59	F
9	Tom	19	15	24	58	F
7	Georgin a	22	12	22	56	F
4	Duke	13	21	20	54	F

7.**Find pass percentage of the class for mathematics.**

SELECT COUNT(maths)*10 as pass_persentage_maths from student where maths>=25

PASS_PERCENTAGE_MATHS
60

8..**Find the overall pass percentage for all class.**

SELECT COUNT(RESULT)*10 AS pass_percentage from student where result='P'

PASS_PERCENTAGE
20

9..**Find the class average.**

SELECT sum(totalmark)/10 AS class_avg from student

CLASS_AVG
50.7

10.**Find the total number of students who have got a Pass.**

SELECT count(result) from student where result='

COUNT(RESULT)
2

RESULT

The query was executed successfully and output was obtained.

EXPERIMENT NO: 7

IMPLEMENTATION OF ORDER BY, GROUP BY&HAVING CLAUSE

AIM:

To understand and implement Order By, Group By&Having clause in SQL

Amazon is one of the largest online stores operating in the United States of America. They are maintaining four tables in their database. The Items table, Customers table, Orders table and Delivery table. Each of these tables contains the following attributes:

Items: - **itemid (primary key)**
Itemname(type =varchar(50))
category
Price
Instock (type=int, greater than or equal to zero)

Customer:- **custid(primary key)**
Custname
Address
State

Orders:- **ordered(primary key)**
Custid
Itemid(refers to item id of items table)
Quantity(type=int)
Orderdate(type=date)

Delivery:- **deliveryid(primary key)**
Custid (refers to custid in customer table)
Orderid(refers to ordered in order table)

Create the above tables and populate them with appropriate data.

CREATE TABLE(itemidint primary key,Itemnamevarchar(50),category varchar(20),Price int,Instockint check(instock>=0))

```
Table created.
```

CREATE TABLE customers(custidint primary key,Custnamevarchar(20),Address varchar(20),state varchar(20))

```
Table created.
```

insert into item values(4,'helmet' CREATE TABLEorders(orderidint primary key,custidint,Itemidint,Quantityint,Orderdate date)

```
Table created.
```

CREATE TABLEdelivery(deliveryidint primary key,Custidint,Orderidint)

```
Table created.
```

ALTER TABLE Orders
ADD FOREIGN KEY (itemid) REFERENCES
item(itemid) ADD FOREIGN KEY (custid)
REFERENCES customers(custid)

```
Table altered.
```

ALTER TABLE delivery
ADD FOREIGN KEY (custid) REFERENCES
customers(custid) ADD FOREIGN KEY (orderid)
REFERENCES orders(orderid)

```
Table altered.
```

insert into item values(1,'batman','toys',10,2)

```
1 row(s) inserted.
```

insert into item values(2,'laptop','electronics',699,5)

```
1 row(s) inserted.
```

insert into item values(3,'galaxy s4','electronics',500,15)

```
1 row(s) inserted.
```

,'vehicles',111,20)

```
1 row(s) inserted.
```

insert into item values(5,'sony z5 premium','electronics',5005,1)

```
1 row(s) inserted.
```

insert into customers values(111,'elvin','202 jai street','delhi')

```
1 row(s) inserted.
```

insert into customers values(112,'patrick','street 1 harinagar','chennai')

```
1 row(s) inserted.
```

insert into customers values(113,'soman','puthumana p.o','kerala')

```
1 row(s) inserted.
```

insert into customers values(114,'jaise','kottarakara','kerala')

```
1 row(s) inserted.
```

insert into customers values(115,'mickey','juhu','mumbai')

```
1 row(s) inserted.
```

insert into orders values(911,111,1,50,'11-oct-2014')

1 row(s) inserted.

insert into orders values(912,113,3,25,'29-jan-2012')

1 row(s) inserted.

insert into orders values(913,115,5,33,'16-may-2013')

1 row(s) inserted.

insert into orders values(914,114,4,35,'22-dec-2014')

```
1 row(s) inserted.
```

insert into delivery values(667,115,914)

```
1 row(s) inserted.
```

insert into delivery values(669,111,911)

```
1 row(s) inserted.
```

insert into delivery values(670,113,912)

```
1 row(s) inserted.
```

1.Find the details of all customers grouped bystate

SELECT * FROM customers ORDER BY state

CUSTID	CUSTNAME	ADDRESS	STATE
112	patrick	street 1 harinagar	chennai
111	elvin	202 jai street	delhi
113	soman	puthumanap.o	kerala
114	jaise	kottarakara	kerala
115	mickey	Juhu	mumbai

2. Display the details of all items grouped by category and having a price greater than average price of all items.

SELECT * FROM item WHERE price>(select avg(price) from item) ORDER BY category

ITEMID	ITEMNAME	CATEGORY	PRICE	INSTOCK
5	sony z5 premium	electronics	5005	1

RESULT

The query was executed successfully and output was obtained.

EXPERIMENT NO: 8

IMPLEMENTATION OF SET OPERATORS NESTED QUERIES ,AND JOIN QUERIES

AIM:

To get introduced to

-UNION
-INTERSECTION
-MINUS
- JOIN
- NESTED QUERIES
- GROUP BY & HAVING

JOINS

A **join** is a query that combines rows from two or more tables, views, or materialized views ("snapshots"). Oracle performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables. If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

Join Conditions

Most join queries contain WHERE clause conditions that compare two columns, each from a different table. Such a condition is called a **join condition**. To execute a join, Oracle combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not also appear in the select list.

To execute a join of three or more tables, Oracle first joins two of the tables based on the join conditions comparing their columns and then joins the result to another table based on join conditions containing columns of the joined tables and the new table. Oracle continues this process until all tables are joined into the result. The

optimizer determines the order in which Oracle joins tables based on the join conditions, indexes on the tables, and, in the case of the cost-based optimization approach, statistics for the tables. In addition to join conditions, the WHERE clause of a join query can also contain other conditions that refer to columns of only one table. These conditions can further restrict the rows returned by the join query.

Equijoins

An **equijoin** is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns. Depending on the internal algorithm the optimizer chooses to execute the join, the total size of the columns in the equijoin condition in a single table may be limited to the size of a data block minus some overhead. The size of a data block is specified by the initialization parameter DB_BLOCK_SIZE.

Self Joins

A **self join** is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition. To perform a self join, Oracle combines and returns rows of the table that satisfy the join condition.

Cartesian Products

If two tables in a join query have no join condition, Oracle returns their **Cartesian product**. Oracle combines each row of one table with each row of the other. A Cartesian product always generates many rows and is rarely useful. For example, the Cartesian product of two tables, each with 100 rows, has 10,000 rows. Always include a join condition unless you specifically need a Cartesian product. If a query joins three or more tables and you do not specify a join condition for a specific pair, the optimizer may choose a join order that avoids producing an intermediate Cartesian product.

Outer Joins

An outer join extends the result of a simple join. An **outer join** returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition. Such rows are not returned by a simple join. To write a query that performs an outer join of tables A and B and returns all rows from A, apply the outer join operator (+) to all columns of B in the join condition. For all rows in A that have no matching rows in B, Oracle returns NULL for any select list expressions containing columns of B.

Outer join queries are subject to the following rules and restrictions:

- The (+) operator can appear only in the WHERE clause or, in the context of left- correlation (that is, when specifying the TABLE clause) in the FROM clause, and can be applied only to a column of a table or view.
- If A and B are joined by multiple join conditions, you must use the (+) operator in all of these conditions. If you do not, Oracle will return only the rows resulting from a simple join, but without a warning or error to advise you that you do not have the results of an outer join.
- The (+) operator can be applied only to a column, not to an arbitrary expression. However, an arbitrary expression can contain a column marked with the (+) operator.
- A condition containing the (+) operator cannot be combined with another condition using the OR logical operator.
- A condition cannot use the IN comparison operator to compare a column marked with the (+) operator with an expression.
- A condition cannot compare any column marked with the (+) operator with a subquery.

If the WHERE clause contains a condition that compares a column from table B with a constant, the (+) operator must be applied to the column so that Oracle returns the rows from table A for which it has generated NULLs for this column. Otherwise Oracle will return only the results of a simple join.

In a query that performs outer joins of more than two pairs of tables, a single table can be the NULL-generated table for only one other table. For this reason, you cannot apply the (+) operator to columns of B in the join condition for A and B and the join condition for B and C

Set Operators:

Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries. Table lists SQL set operators.

Operator	Returns
UNION	All rows selected by either query.
UNION ALL	All rows selected by either query, including all duplicates.
INTERSECT	All distinct rows selected by both queries.
MINUS	All distinct rows selected by the first query but not the second.

All set operators have equal precedence. If a SQL statement contains multiple set operators, Oracle evaluates them from the left to right if no parentheses explicitly specify another order.

The corresponding expressions in the select lists of the component queries of a compound query must match in number and datatype. If component queries select character data, the datatype of the return values are determined as follows:

- If both queries select values of datatype CHAR, the returned values have datatype CHAR.

- If either or both of the queries select values of datatype VARCHAR2, the returned values have datatype VARCHAR2.

Set Membership (IN & NOT IN):

NOT IN Operator

If any item in the list following a NOT IN operation is null, all rows evaluate to UNKNOWN (and no rows are returned). For example, the following statement returns the string 'TRUE' for each row:

```
SELECT 'TRUE'
FROM emp
WHERE deptno NOT IN (5,15);
```

However, the following statement returns no

rows:

```
SELECT 'TRUE'
FROM emp
WHERE deptno NOT IN (5,15,null);
```

The above example returns no rows because the WHERE clause condition evaluates

to: deptno != 5 AND deptno != 15 AND deptno != null

Because all conditions that compare a null result in a null, the entire expression results in a null. This behavior can easily be overlooked, especially when the NOT IN operator references a subquery.

NESTED QUERIES

Sub query:

If a sql statement contains another sql statement then the sql statement which is inside another sql statement is called Subquery. It is also known as nested query. The Sql Statement which contains the other sql statement is called Parent Statement.

Nested Sub query:

If a Subquery contains another subquery, then the subquery inside another subquery is called nested subquery.

Correlated Subquery:

If the outcome of a subquery is depends on the value of a column of its parent query table then the Sub query is called Correlated Subquery.

QUESTION

Amazon is one of the largest online stores operating in the United States of America. They are maintaining four tables in their database. The Items table, Customers table, Orders table and Delivery table. Each of these tables contains the following attributes:

Items: -	**itemid (primary key)**
	Itemname(type =varchar(50))
	category
	Price
	Instock (type=int, greater than or equal to zero)
Customers:-	**custid (primary key)**
	Custname
	Address
	State
Orders:-	**orderid (primary key)**
	Itemid(refers to itemid of Items table)
	Quantity (type=int)
	Orderdate (type=date)
Delivery:-	**deliveryid (primary key)**
	Custid (refers to custid in customers table) Orderid (refers to ordered in orders table)

Create the above tables and populate them with appropriate data.

CREATE TABLE(itemidint primary key,Itemnamevarchar(50),category varchar(20),Price int,Instockint check(instock>=0))

```
Table created.
```

CREATE TABLE customers(custidint primary key,Custnamevarchar(20),Address varchar(20),state varchar(20))

```
Table created.
```

CREATE TABLEorders(orderidint primary key,custidint,Itemidint,Quantityint,Orderdate date)

```
Table created.
```

CREATE TABLEdelivery(deliveryidint primary key,Custidint,Orderidint)

```
Table created.
```

ALTER TABLE Orders
ADD FOREIGN KEY (itemid) REFERENCES
item(itemid) ADD FOREIGN KEY (custid)
REFERENCES customers(custid)

```
Table altered.
```

ALTER TABLE delivery
ADD FOREIGN KEY (custid) REFERENCES
customers(custid) ADD FOREIGN KEY (orderid)
REFERENCES orders(orderid)

```
Table altered.
```

insert into item values(1,'batman','toys',10,2)

```
1 row(s) inserted.
```

insert into item values(2,'laptop','electronics',699,5)

```
1 row(s) inserted.
```

insert into item values(3,'galaxy s4','electronics',500,15)

```
1 row(s) inserted.
```

insert into item values(4,'helmet','vehicles',111,20)

```
1 row(s) inserted.
```

insert into item values(5,'sony z5 premium','electronics',5005,1)

```
1 row(s) inserted.
```

insert into customers values(111,'elvin','202 jai street','delhi')

```
1 row(s) inserted.
```

insert into customers values(112,'patrick','street 1 harinagar','chennai')

```
1 row(s) inserted.
```

insert into customers values(113,'soman','puthumana p.o','kerala')

```
1 row(s) inserted.
```

insert into customers values(114,'jaise','kottarakara','kerala')

```
1 row(s) inserted.
```

insert into customers values(115,'mickey','juhu','mumbai')

```
1 row(s) inserted.
```

insert into orders values(911,111,1,'11-oct-2014')

```
1 row(s) inserted.
```

insert into orders values(912,113,3,'29-jan-2012')

```
1 row(s) inserted.
```

insert into orders values(913,115,5,'16-may-2013')

```
1 row(s) inserted.
```

insert into orders values(914,114,4,'22-dec-2014')

```
1 row(s) inserted.
```

insert into delivery values(667,115,914)

```
1 row(s) inserted.
```

insert into delivery values(669,111,911)

```
1 row(s) inserted.
```

insert into delivery values(670,113,912)

```
1 row(s) inserted.
```

1.List the details of all customers who have placed an order

SELECT * FROM customers WHERE custidIN(select custid from orders)

CUSTID	CUSTNAME	ADDRESS	STATE
111	elvin	202 jai street	delhi
113	soman	puthumanap.o	kerala
115	mickey	juhu	mumbai
114	jaise	kottarakara	kerala

2.List the details of all customers whose orders have been delivered

SELECT* FROM customers WHEREcustid IN (select custid from delivery)

CUSTID	CUSTNAME	ADDRESS	STATE
115	mickey	juhu	mumbai
111	elvin	202 jai street	delhi
113	soman	puthumanap.o	kerala

3. Find the orderdate for all customers whose name starts in the letter 'J'

SELECT orderdate FROM orders WHEREcustidin(select custid from customers where custname LIKE 'j%')

ORDERDATE
22-DEC-14

4.Display the name and price of all items bought by the customer 'Mickey'

SELECT itemname,price FROM item where itemid IN(select itemid from orders where custid in(select custid from customers where custname='mickey'))

ITEMNAME	PRICE
sony z5 premium	5005

5.List the details of all customers who have placed an order after January 2013 and not received delivery of items.

SELECT * FROM customers WHEREcustidIN(select custid from orders where orderdate>'31- jan-2013' AND orderid NOT IN(select orderid from delivery))

CUSTID	CUSTNAME	ADDRESS	STATE
114	jaise	kottarakara	kerala

6.Find the itemid of items which has either been ordered or not delivered. (Use SET UNION)

SELECT itemid FROM orders UNION SELECT itemid FROM orders where custidIN(select custid from delivery)

ITEMID
1
3
4
5

7.Find the name of all customers who have placed an order and have their orders delivered.(Use SET INTERSECTION)

SELECT custname FROM customers WHERE custidIN(select custid from orders)

INTERSECT SELECT custname FROM customers WHERE custid IN(select custid from delivery)

CUSTNAME
Elvin
Mickey
Soman

8.Find the custname of all customers who have placed an order but not having their orders delivered. (Use SET MINUS)

SELECT custname FROM customers WHERE custidIN(select custid from orders) MINUS select custname from customers where custid IN(select custid from delivery)

CUSTNAME
jaise

9.Find the name of the customer who has placed the most number of orders.

SELECT * FROM customers WHERE custidIN(select custid from orders where quantity = (select max(quantity) from orders))

CUSTID	CUSTNAME	ADDRESS	STATE
114	jaise	kottarakara	kerala

10.Find the details of all customers who have purchased items exceeding a price of 5000 $.

SELECT * FROM customers WHERE custidIN(select custid from orders where itemid IN(select itemid from item where price>5000))

CUSTID	CUSTNAME	ADDRESS	STATE
115	mickey	juhu	mumbai

11.Find the name and address of customers who has not ordered a 'Samsung Galaxy S4'

SELECT custname,address FROM customers WHERE custid NOT IN(select custid from orders where itemid IN(select itemid from item where itemname='galaxy s4'))

CUSTNAME	ADDRESS
elvin	202 jai street
patrick	street 1 harinagar
jaise	kottarakara
mickey	juhu

12.Perform Left Outer Join and Right Outer Join on Customers & Orders Table.

SELECT * FROM customers LEFT JOIN orders ON customers.custid=orders.custid;

CUS TID	CUSTNAME	ADDRESS	STATE	ORDERID	CUSTID	ITEMID	QUANTITY	ORDER DATE
111	elvin	202 jai street	delhi	911	111	1	2	11-OCT-14
113	soman	puthumanap.o	kerala	912	113	3	1	29-JAN-12
115	mickey	juhu	mumbai	913	115	5	1	16-MAY-13
114	jaise	kottarakara	kerala	914	114	4	3	22-DEC-14
112	patrick	street 1 harinagar	chennai	-	-	-	-	-

13.Find the details of all customers grouped by state

SELECT * FROM customers ORDER BY state

CUSTID	CUSTNAME	ADDRESS	STATE
112	patrick	street 1 harinagar	chennai
111	elvin	202 jai street	delhi
113	soman	puthumanap.o	kerala
114	jaise	kottarakara	kerala
115	mickey	juhu	mumbai

14.Display the details of all items grouped by category and having a price greater than average price of all items.

SELECT * FROM item WHERE price>(select avg(price) from item) ORDER BY category

ITEMID	ITEMNAME	CATEGORY	PRICE	INSTOCK
5	sony z5 premium	electronics	5005	1

RESULT

The query was executed and output was successfully obtained.

EXPERIMENT NO: 9

TCL COMMANDS

AIM:

Familiarize and practice of SQL TCL commands like Rollback, Commit, Save point.

THEORY:

A transaction is a logical unit of work.

Commit

The transaction changes can be made permanent to a database.

Sql> commit;

Savepoint

Markers to divide a very lengthy transaction to smaller ones.

Used to identify a point in a transaction to which we can later rollback.

Sql> savepoint savepoint_id1;

Rollback

Used to undo the work done in the current transaction.

Rollback a transaction to a save point so that the SQL statements after the save point are rolled back.

Sql> rollback;

Sql> rollback to savepoint_id1;

1. Create a table student with fields name ,rollno and address.

SQL> create table student(name varchar(20),roll number,address varchar(30));

Table created.

2. Insert vales into table and display the details

SQL> insert into student values('arun',13,'aroma');

1 row created.

SQL> insert into student values('kiran',28,'aroma');

1 row created.

SQL> select * from student;

NAME ROLL ADDRESS

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

arun 13 aroma

kiran 28 aroma

SQL> savepoint in_student;

Savepoint created.

3.Rollback to a savepoint and display the details

SQL> insert into student values('ned',35,'winterfell');

1 row created.

SQL> select * from student;

NAME ROLL ADDRESS

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

arun 13 aroma

kiran 28 aroma

ned 35 winterfell

SQL> rollback to in_student;

Rollback complete.

SQL> select * from student;

NAME ROLL ADDRESS

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

arun 13 aroma

kiran 28 aroma

RESULT

The query was executed and output was successfully obtained.

EXPERIMENT NO: 10

DCL COMMANDS

AIM:

Familiarize and Practice of SQL DCL commands for granting and revoking user privileges.

THEORY:

1. Create a table student with fields name ,rollno and address.

SQL> create table student(name varchar(20),roll number,address varchar(30));

Table created.

2.Insert vales into table and display the details

SQL> insert into student values('arun',13,'aroma');

1 row created.

SQL> insert into student values('kiran',28,'aroma');

1 row created.

SQL> insert into student values('ned',35,'winterfell');

1 row created.

SQL> select * from student;

NAME	ROLL	ADDRESS
arun	13	aroma
kiran	28	aroma
ned	35	winterfell

3.Grant dba privilege to a user

SQL> create user puser identified by abc;

User created.

4.Grant select privilege on students

SQL> grant dba to puser;

Grant succeeded.

5.Revoke select on students from privileged user

SQL> revoke select on order1 from puser;

Revoke succeeded.

RESULT

The query was executed and output was successfully obtained.

EXPERIMENT NO: 11

CREATION OF VIEWS

AIM:

To understand and implement Creation of Views and Assertionsin SQL.

1.Create a View named sales_staff to hold the details of all staff working in sales Department

CREATE VIEW sales_staffas(SELECT *FROM staff WHEREdept='Sales')

```
View created.
```

SELECT *FROM sales_staff

STAFF_ID	STAFF_NAME	DEPT	AGE	SALARY
1	Sera	Sales	25	20000
3	Jane	Sales	28	25000

2.Drop table branch. Create another table named branch and name all the constraints as givenbelow:

Constraint name	Column	Constraint
Pk	**branch_id**	**Primary key**
Df	**branch_name**	**Default :'New Delhi'**
Fk	**bankid**	**Foreign key/References**

DROP TABLE branch

```
Table droped.
```

CREATE TABLE Branch(branch_idint,branchnamevarchar(15)DEFAULT('New Delhi'),bankidvarchar(3))

```
Table created.
```

ALTER TABLE Branch ADD CONSTRAINT pk PRIMARY KEY(branch_id)

```
Table altered.
```

ALTER TABLE Branch ADD CONSTRAINT fk FOREINGN KEY(bankid) REFERENCES bank(bankcode)

```
Table altered.
```

i)Delete the default constraint in thetable

ALTER TABLE Branch ALTER COLUMN branch_name DROP DEFAULT

```
Table altered.
```

ii)Delete the primary keyconstraint

ALTER TABLE Branch DROP CONSTRAINT pk

```
Table altered.
```

3.Update the view sales_staff to include the details of staff belonging to sales department whose salary is greater than20000.

CREATE OR REPLACE sales_staff as SELECT *FROM Staff WHERESalary>20000

```
View created.
```

SELECT *FROM sales_staff

STAFF_ID	STAFF_NAME	DEPT	AGE	SALARY
1	John	Purchasing	24	30000
2	Sera	Sales	25	20000
3	Jane	Sales	28	25000

4.Delete the viewsales_staff.

DROP VIEWsales_staff

```
View droped.
```

RESULT

The query was executed and output was successfully obtained.

EXPERIMENT NO: 12

IMPLEMENTATION OF VARIOUS CONTROL STRUCTURES USING PL/SQL

AIM:

To understand and implement various control structures using PL/SQL

Write PL/SQL programs for the following:

1. To perform addition, subtraction, multiplication and division on twonumbers.

```
DECLARE
        aint;
        bint;
BEGIN
       a:=:a;
       b:=:b;
       dbms_output.put_line('sum is'||(a+b));
       dbms_output.put_line('difference is'||(a-b));
       dbms_output.put_line('product is'||(a*b));
       dbms_output.put_line('qoutient is'||(a/b));
       END
```

OUTPUT:

```
     A:30
     B:5

sum is35
difference is25
product is150
quotient is6

     Statement processed.
```

2. To print the first 'n' primenumbers

```
DECLARE

        iint;
       j int;
       nint;
       numint;
```

```
        cntint
        flag int;

BEGIN
        flag:=0;
        int:=0;
        num:=2;
        whileint<20
        loop
        flag:=0;
        i:=num;
           for j in
        2..i/2 loop
        If mod(i,j)=0 then
        flag:=1;
        end if;
        end loop;
        if flag=0 then
        int:int+1;
        dbms_output.put_line(i);
        end if;
        num:=num+1;
        end loop;
        END;
```

OUTPUT:

```
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71


Statement processed.
```

3.Display the Fibonacci series upto ‘n’terms

```
DECLARE
    aint;
    bint;
    cint;
    cntint;
    i int;
BEGIN
    i:=:I;
    int:=1;
    a:=0;
    b:=1;
    dbms_output,put_line(a);
    dbms_output,put_line(b);
    for int in 1..i-2
    loop
    c:=a+b;

      dbms_output,put_line(c);a:=b;
    b:=c;
    end loop;
END
```

4. Create a table named student grade with the given attributes: roll, name, mark1,mark2,mark3, grade.Read the roll,name and marks from the user. Calculate the grade of the student and insert a tuple into the using PL/SQL.(Grade=’PASS’ if AVG>40,Grade=’FAIL’ otherwise)

```
CREATE TABLEstudent_grade(roll int,name varchar2(10),mark 1 int,mark2
int,mark3 int,gradevarchar(5));

Table created.

DECLARE
      totalint;
      avrg float;
      roll int;
      mark1 int;
      mark2 int;
      mark3 int;
      grade varchar2(5);
      name varchar2(10);
```

```
BEGIN
        roll:=:roll;
        name:=:name:
        mark1:=:mark1;mark2:=:mark2;
        mark3:=:mark3;
        total:=mark1+mark2+mark3;
        avrg:=total/3;
        ifavrg>40 then
        grade:='PASS';else
        grade:='FAIL';end if;
        insert into student_grade values(roll,name,mark1,mark2,mark3,grade);
        END;
        select* from student_grade
```

ROLL	NAME	MARK1	MARK2	MARK3	GRADE
1	anu	50	45	48	PASS
2	manu	50	50	50	PASS
3	manu	35	40	40	FAIL

5. **Create table circle_area (rad,area). For radius 5,10,15,20 &25., find the area andinsert the corresponding values into the table by using loop structure inPL/SQL.**

```
create table circle_area(radius int,area float(5));


DECLARE
        radiusint;
        iint;
       area float;
BEGIN

        radius:=0;
        for i in 1.5 loop
        radius:=radius+5;
        area:=3.14*power(radius,2);
        INSERT INTO circle_areavalues(radius,area);
        end loop;
END;
        SELECT*from circle_area;
```

RADIUS	AREA
5	79
10	310
15	710
20	1300
25	2000

6.Use an array to store the names, marks of 10 students in a class. Using Loop structures in PL/SQL insert the ten tuples to a table namedstud

```
CREATE TABLEstud(names varchar2(10),marks int);

DECLARE
        typestdnames is varray(10) of varchar2(10);
        typestdmarks is varray(10) of int;
        namesstdnames;
        marksstdmarks;
        iint;

BEGIN
    names:=stdnames('ARUN','AMAL','PETER','JOSE','ANNIE','MARY','JOSEPH','MARK'
     ,'MIDHUN','KEVIN');
       marks:=stdmarks(25,76,43,45,67,57,97,56,89,08);
       for i in 1..10 loop
       INSERT INTO stud values(names(i),marks(i));
       end loop;

END;
       SELECT * from stud
```

NAMES	MARKS
ARUN	25
AMAL	76
PETER	43
JOSE	45
ANNIE	67
MARY	57
JOSEPH	97
MARK	56
MIDHUN	89
KEVIN	8

7.**Create a sequence using PL/SQL. Use this sequence to generate the primary key values for a table named class_cse with attributes roll,name and phone. Insert some tuples using PL/SQLprogramming.**

```
CREATE TABLE class_cse(roll int,name varchar2(10),phone varchar(20));
create sequence seql
start with 1
increment by 1
maxvalue 10
DECLARE
        typestdnameis varray(10) of varchar2(10);
        typestdphoneis varray(10) of varchar2(20);
```

```
    namestdname;
    phonestphone

BEGIN
     name:=stdname('ARUN','AMAL','PETER','JOSE','ANNIE');
     phone:=stdphone('0482-239091','0484-234562','0485-11234','0489-43617','0481-23145'
     for i in 1..5 loop
     INSERT INTOclass_csevalues(seql.nextval,name(i),phone(i));
     end loop;

END;
      SELECT * from class_cse;
```

ROLL	NAME	PHONE
1	ARUN	0482-239091
2	AMAL	0484-234562
3	PETER	0485-11234
4	JOSE	0489-43617
5	ANNIE	0481-23145

RESULT

The query was executed and output was successfully obtained.

EXPERIMENT NO: 13

CREATION OF PROCEDURES, TRIGGERS AND FUNCTIONS

AIM:

To study about procedure and functions

THEORETICAL BACKGROUND:

PL/SQL Procedures and Functions

Procedure

- A **stored procedure** or in simple a **proc** is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.
- A procedure has a header and a body.
    - The header consists of the name of the procedure and the parameters or variables passed to the procedure.
    - The body consists of
        - declaration section
        - execution section
        - exception section
- A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.
- A procedure may or may not return any value.
- Syntax to create a procedure is:

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of
parameters]
IS
  Declaration section
BEGIN
  Execution section
EXCEPTION
 Exception section
END;
```

    - **IS -** marks the beginning of the body of the procedure
    - By using CREATE OR REPLACE together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.
    - backward slash '/' at the end of the program indicates the oracle engine that the PL/SQL program has ended and it can begin processing the statements.
- Executing a Stored Procedure

- There are two ways to execute a procedure.
  1) From the SQL prompt.

     EXECUTE [or EXEC] procedure_name;

  2) Within another procedure – simply use the procedure name.

     procedure_name;

- To drop a procedure

  DROP PROCEDURE procedurename;

Write a procedure that that takes two numbers as parameters and display the multiplication table of first parameter till second.

```
SQL> create or replace procedure mult_table(a number,b number) is
  mul number;
  begin
 for i in 1. .b loop
  mul:=a*i;
 dbms_output.put_line(to_char(a)||'*'||to_char(i)||'='||to_char(mul));
 end loop;
 end mult_table;
  /
```

Procedure created.

```
SQL> set serveroutput on;
SQL> declare
 2  c number;
 3  d number;
 4  begin
 5  c:=&c;
```

```
  6  d:=&d;
  7  mult_table(c,d);
  8  end;
  9  /
```

OUTPUT

```
Enter value for c: 5
old   5: c:=&c;
new   5: c:=5;
Enter value for d: 4
old   6: d:=&d;
new   6: d:=4;
5*1=5
5*2=10
5*3=15
5*4=20
```

PL/SQL procedure successfully completed

Commit complete.

Consider an employee table.Write a procedure raise salary .It accepts an employee number and increases the salary amount.It uses employee number to find current salary from employee table and update it

SQL> select * from employee;

EMPID	EMPNAME	SALARY	PID
1001	Ravi	25500	2
1002	Rajesh	30000	1
1003	Ramesh	22500	3
1004	Ratheesh	16000	5

```
SQL> create or replace procedure raisesal(empl number,amt number) is
  begin
 update employee set salary=salary+amt where empid=empl;
  end;
  /
Procedure created
SQL> set serveroutput on;
SQL> declare
 empn number;
  amt number;
  begin
 dbms_output.put_line('Enter employee number');
 empn:=&empn;
 dbms_output.put_line('Enter salary increment');
  amt:=&amt;
  raisesal(empn,amt);
dbms_output.put_line('Salary of '||to_char(empn)||'increased by '||to_char(amt));
 end;
12  /
```

OUTPUT

```
Enter value for empn: 1001
old   6: empn:=&empn;
new   6: empn:=1001;
Enter value for amt: 1000
old   8: amt:=&amt;
new   8: amt:=1000;
```

Enter employee number

Enter salary increment

Salary of 1001increased by 1000

PL/SQL procedure successfully completed.

SQL> select * from employee;

EMPID	EMPNAME	SALARY	PID
1001	Ravi	26500	2
1002	Rajesh	30000	1
1003	Ramesh	22500	3
1004	Ratheesh	16000	5

RESULT

The query was executed and output was successfully obtained.

EXPERIMENT NO: 14
CREATION OF PACKAGES

AIM:
To solve queries using packages

THEORETICAL BACKGROUND:

- PL/SQL package is a group of related stored functions, procedures, types, cursors and etc.
- PL/SQL package is like a library once written stored in the Oracle database and can be used by many applications.
- A package has two parts:
    - A package specification is the public interface of your applications.
    - A package body contains the code that implements the package specification.

Package Specification

- The package specification is required when you create a new package.
- The package specification lists all the objects which are publicly accessible from other applications.
- Syntax:

```
CREATE [OR REPLACE] PACKAGE package_name

{ IS | AS }

  [definitions OF PUBLIC TYPES

  ,declarations OF PUBLIC variables, types and objects

  ,declarations OF exceptions

   ,declarations OF cursors, procedures and functions

  ,headers OF procedures and functions]

END [package_name]
```

Package Body

- PL/SQL package body contains all the code that implements stored functions, procedures and cursors listed in the package specification.
    - Syntax

```
CREATE [OR REPLACE] PACKAGE BODY package_name
  { IS | AS }

    [definitions OF PRIVATE TYPEs
```

```
    ,declarations OF PRIVATE variables, types and objects
    ,full definitions OF cursors
    ,full definitions OF procedures  and functions]
  [BEGIN
    sequence_of_statements
   [EXCEPTION
    exception_handlers ] ]
  END [package_name];
```

Referencing PL/SQL Package Elements

- Syntax: *package_name.package_element*

Questions

1.Create a table epack.Create a package payroll for calculating the salary of each employee.By using a procedure enter the values of name, no dateof birth

```
SQL> create package custsal as
  2  procedure find_sal(c_id customers.id%type);
  3  end custsal;
SQL> /
Package created.

SQL> create or replace package body custsal as
  2  procedure find_sal(c_id customers.id%type) is
  3  c_sal customers.salary%type;
  4  begin
  5  select salary into c_sal from customers where id=c_id;
  6  dbms_output.put_line('Salary: '||c_sal);
```

```
 7  end find_sal;
 8  end custsal;
 9  /
```

Package body created.

```
SQL> set serveroutput on;
SQL> execute custsal.find_sal(1);
```

Salary: 2500

PL/SQL procedure successfully completed.

2.SQL> create table epack(name varchar2(20),no number,dob date,bp number,hra number,ta number,da number,ded number,gp number,np number);

Table created.

```
SQL> create or replace package epackage3 as
 2  procedure data_entry(name2 epack.name%type,no2 epack.no%type,dob2 epack.dob%type,bp2 epack.bp%type);
 3  procedure pay_bill(no3 epack.no%type);
 4  end epackage3;
 5  /
```

Package created.

```
 SQL> create or replace package body epackage3 as
 2  procedure data_entry(name2 epack.name%type,no2 epack.no%type,dob2 epack.dob%type,bp2 epack.bp%type) is
 3  begin
 4  insert into epack(name,no,dob,bp) values(name2,no2,dob2,bp2);
 5  end data_entry;
 6  procedure pay_bill(no3 epack.no%type) is
```

```
 7   hra3 epack.hra%type;
 8   ta3 epack.ta%type;
 9   da3 epack.da%type;
10   ded3 epack.ded%type;
11   gp3 epack.gp%type;
12   np3 epack.np%type;
13   bp3 epack.bp%type;
14   begin
15   select bp into bp3 from epack where no=no3;
16   if bp3<1500 then
17   hra3:=bp3*0.1;
18   ta3:=bp3*0.8;
19   da3:=bp3*0.8;
20   ded3:=bp3*0.01;
21   else
22   hra3:=bp3*0.12;
23   ta3:=bp3*0.09;
24   da3:=bp3*0.09;
25   ded3:=bp3*0.9;
26   end if;
27   gp3:=bp3+hra3+ta3+da3;
28   np3:=gp3-ded3;
29  insert into epack(hra,ta,da,ded,gp,np) values(hra3,ta3,da3,ded3,gp3,np3);
30   end pay_bill;
31*  end epackage3;
```

```
SQL> execute epackage3.data_entry('kiran',201,'25-FEB-2000',2500);

PL/SQL procedure successfully completed.


SQL> select * from epack;

NAME                    NO DOB            BP       HRA        TA
-------------------- ---------- --------- ---------- ---------- ----------
      DA       DED        GP        NP
---------- ---------- ---------- ----------
kiran                  201 25-FEB-00       2500


SQL> execute epackage3.pay_bill(201);

PL/SQL procedure successfully completed.


SQL> select * from epack;

NAME                    NO DOB            BP       HRA        TA
-------------------- ---------- --------- ---------- ---------- ----------
      DA       DED        GP        NP
---------- ---------- ---------- ----------
kiran                  201 25-FEB-00       2500    300        225
```

RESULT

The query was executed and output was successfully obtained.

EXPERIMENT NO: 15

CREATION OF CURSORS

AIM:

To study PL/SQL-CURSOR Programming

Oracle creates a memory area, known as context area or work area, for processing an SQL statement, which contains all information needed for processing the statement.A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. T he set of rows the cursor holds is referred to as the active set.

T here are two types of cursors:

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed. Programmers cannot control the implicit cursors and the information in it.

- These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed.
- They are also created when a SELECT statement that returns just one row is executed.
- Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations.
- The cursor attributes available are %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.

Explicit Cursors

Explicit cursors are programmer defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

Working with an explicit cursor involves four steps:

Declaring the cursor for initializing in the memory

Syntax: CURSOR cursor_name IS select_statement;

Opening the cursor for allocating memory

Syntax: OPEN cursor_name;

Fetching the cursor for retrieving data

Syntax: FETCH cursor_name INTO record_name;

OR

FETCH cursor_name INTO variable_list;

Closing the cursor to release allocated memory

Syntax: CLOSE cursor_name;

Questions

1. Create a table employee with fields employee number,basic pay,allowance and total salary.Set total salary to zero.Write a pl/sql program to calculate total salary.Total salary =basic pay +1000

create table emply(empno number,basicpay number,allowance number,total number default 0);

Table created.

insert into emply values(&empno,&basicpay,&allowance,0);

Enter value for empno: 101

Enter value for basicpay: 35000

Enter value for allowance: 5000

old 1: insert into emply values(&empno,&basicpay,&allowance,0)

new 1: insert into emply values(101,35000,5000,0)

1 row created.

insert into emply values(&empno,&basicpay,&allowance,0);

Enter value for empno: 102

Enter value for basicpay: 42000

Enter value for allowance: 3000

old 1: insert into emply values(&empno,&basicpay,&allowance,0)

```
new   1: insert into emply values(102,42000,3000,0)

1 row created.

insert into emply values(&empno,&basicpay,&allowance,0);
Enter value for empno: 103
Enter value for basicpay: 52000
Enter value for allowance: 4000
old   1: insert into emply values(&empno,&basicpay,&allowance,0)
new   1: insert into emply values(103,52000,4000,0)

1 row created.
```

2.

```
declare
  2 emply.empno%type;
  3  basic emply.basicpay%type;
  4  allow emply.allowance%type;
  5  cursor emp_cur is
  6  select empno,basicpay,allowance
  7  from emply;
  8  begin
  9  if not emp_cur%isopen then
 10  open emp_cur;
 11  end if;
 12  loop
 13  fetch emp_cur
```

```
 14  into id,basic,allow;
 15  if id>10 then
 16  update emply set total=basic+allow where id=empno;
 17  else
 18  update emply set total=basic+1000 where id=empno;
 19  end if;
 20  exit when not emp_cur%found;
 21  dbms_output.put_line(id||' '||basic||' '||allow);
 22  end loop;
 23  close emp_cur;
 24  end;
 25  /
101 35000 5000
102 42000 3000
103 52000 4000

PL/SQL procedure successfully completed.

SQL> select * from emply;

    EMPNO   BASICPAY  ALLOWANCE      TOTAL
---------- ---------- ---------- ----------
       101      35000       5000      40000
       102      42000       3000      45000
       103      52000       4000      56000
```

RESULT

The query was executed and output was successfully obtained.

EXPERIMENT NO: 16

CREATION OF PL/SQL BLOCKS FOR EXCEPTIONS

AIM:

To study PL/SQL Exceptions

THEORETICAL BACKGROUND:

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling.

1) Exception Handling we can test the code and avoid it from exiting abruptly.

2) When an exception occurs a messages which explains its cause is recieved.

3) PL/SQL Exception message consists of three parts.

   1) Type of Exception

   2) An Error Code

   3) A message

4) Syntax for coding the exception section

```
DECLARE
  Declaration section
BEGIN
  Exception section
EXCEPTION
WHEN ex_name1 THEN
   -Error handling statements
WHEN ex_name2 THEN
   -Error handling statements
```

WHEN Others THEN

-Error handling statements

END;

- Types of Exception.

o There are 3 types of Exceptions.

a) Pr-Defined Exceptions

- System exceptions and they are pre-defined

b) User-defined Exceptions

•we can explicity define exceptions based on business rules.

Steps to be followed to use user-defined exceptions:

• They should be explicitly declared in the declaration section.

• They should be explicitly raised in the Execution Section.

• They should be handled by referencing the user-defined exception name in the exception section.

Questions

1. Create table product with fields productid,product name,product description and available quantity.

2. Create table orders with fields orderid,productid and order quantity.

3 .Insert values to both tables.

4. Create a pl/sql program to select details of products with id=100.Write exception to handle

a) no data found b)cursor already open

5. Write a PL/SQL program to find out product name and number of products ordered. If total number of items is greater than 20 then display discount applicable, else display discount not applicable.

1.

SQL> create table product(pid number,pname varchar(20),pdesc varchar(20),availqty number);

Table created.

2.

SQL> create table orders(oid number,pid number,orderqty number)

Table created.

3.

SQL> insert into product values(101,'p1','abc',50)

1 row created.

SQL> insert into product values(102,'p2','bcd',45);

1 row created.

SQL> select * from product;

PID	PNAME	PDESC	AVAILQTY
101	p1	abc	50
102	p2	bcd	45
103	p3	cde	100

SQL> select * from orders;

OID	PID	ORDERQTY
1	101	10
2	101	20

3 102 10

4 103 10

4.

```
SQL>
  1  declare
  2  id product.pid%type;
  3  name product.pname%type;
  4  descrip product.pdesc%type;
  5  qty product.availqty%type;
  6  begin
  7  select pid,pname,pdesc,availqty into id,name,descrip,qty from product
  8  where id=100;
  9  exception
 10   when no_data_found then
 11   dbms_output.put_line('no data found');
 12* end;
SQL> /
no data found

PL/SQL procedure successfully completed.
SQL>
  1  declare
  2  id product.pid%type;
  3  name product.pname%type;
```

```
 4  descrip product.pdesc%type;
 5  qty product.availqty%type;
 6  cursor pro_cur is
 7  select pid,pname,pdesc,availqty
 8  from product where id=100;
 9  begin
10  open pro_cur;
11  open pro_cur;
12  loop
13  fetch pro_cur
14  into id,name,descrip,qty;
15  end loop;
16  close pro_cur;
17  exception
18  when cursor_already_open then
19  dbms_output.put_line('Cursor is already open');
20* end;

SQL> /
Cursor is already open


PL/SQL procedure successfully completed.
```

5.

```
SQL> set serveroutput on;
SQL> declare
  2  huge exception;
```

```
 3  small exception;
 4  name product.pname%type;
 5  numb orders.orderqty%type;
 6  cursor prod_cur is
 7  select pname,orderqty
 8  from product,orders where product.pid=orders.pid;
 9  begin
10  if not prod_cur%isopen then
11  open prod_cur;
12  end if;
13  loop
14  fetch prod_cur
15  into name,numb;
16  dbms_output.put_line(name||' '||numb);
17  if numb>19 then
18  raise huge;
19  else
20  raise small;
21  end if;
22  exit when not prod_cur%found;
23  end loop;
24  exception
25  when huge then
26  dbms_output.put_line('Discount applicable');
27  when small then
28  dbms_output.put_line('Discount not applicable');
29  end;
```

30 /

1 10

Discount not applicable

PL/SQL procedure successfully completed.

RESULT

The query was executed and output was successfully obtained.