Ex.No.: 1	CREATION OF BASE TABLE AND DML OPERATIONS
Date: 27/7/24	

AIM:

To creation the Base table and DML operations.

ALGORITHM:

STEP-1: Start.

STEP-2: Create a base Table

Syntax:

CREATE TABLE <table name> (column1 type, column2 type, …);

STEP-3: Describe the Table structure

Syntax:

DESC <table name>

STEP-4: Add a new row to a Table using INSERT statement.

Syntax:

- INSERT INTO <table name> VALUES (value1, value2..);
- INSERT INTO <table name> (column1, column2..)
  VALUES (value1, value2..);
- INSERT INTO <table name>VALUES (&column1,'&column');

STEP-5: Modify the existing rows in the base Table with UPDATE statement.

Syntax:

UPDATE <table name> SET column1=value, column2 = 'value'
WHERE (condition);

STEP-6: Remove the existing rows from the Table using DELETE statement.

Syntax:

DELETE FROM <table name> WHERE <condition>;

STEP-7: Perform a Query using SELECT statement.

Syntax:

SELECT [DISTINCT] {*,<column1,,..>} FROM <table name>
WHERE <condition>;

6. Empty the fourth row of the emp table.

```
DELETE FROM MY-EMPLOYEE
where ID=4;
```

7. Make the data additions permanent.

```
COMMIT;
SELECT * From my-Employee;
```

8. Change the last name of employee 3 to Drexler.

```
UPDATE My-EmployEE
SET last-name = 'Drexler'
where ID=3;
```

9. Change the salary to 1000 for all the employees with a salary less than 900.

```
UPDATE MY-EMPLOYEE
SET last-name = 'Dresler' SET salary=1000 where
salary <90
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

1) Create Table

ID last-name First-name userid salary

2) Insert

```
Insert into MY- EMPLOYEE
  Values (1, 'patel', 'Ralph', 'rpatel', 895)
         (2, 'Dans', 'Betty', 'bdancs', 860);
```

3) SELECT * From MY- EMPLOYEE;

ID	Last-name	Firstname	user-id	salary
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860

4) Concat for next two rows for creating user id.

ID	Last-name	First-name	user-id	salary
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	Chewman	750

5) Delete Betty an Dancs from Id 2 Sa

ID	Last-name	First-name	userid	salary
1	Ralph	Ratel	rpatel	895
2	Ben	Biri	bdancs	860
3	chad	Newman	bbiri	1100
4			Chewman	750.

6) Delete the 4th row

ID	Last-name	First-name	userid	salary
1	Patel	Ralph	rpatel	895
2			bdanes	860
3.	Biri	Ben	bbiri	1100.

7) Data addition permanent:

Commit:

Select * From my-employee:

ID	Last-name	First-name	user-id	salary
1	Patel	Ralph	rpatel	895
2			bdanes	860
3	Biri	Ben	bbiri	1100

8) Last name of employee 3 to Pexler update

my-employee SET Last-name = "Drexler" where ID = 3

ID	Last-name	First-name	userid	salary
1	Patel	Ralph	rpatel	895
2			bdanes	860
3	Drexler	Ben	bbiri	1100.

Result:

Thus we successfully executed the Query in MySQL.

Ex.No.: 2	
Date: 30/7/24	DATA MANIPULATIONS

Create the following tables with the given structure.

The aim to create table & Implement the Queries.

EMPLOYEES TABLE

NAME	NULL?	TYPE
Employee_id	Not null	Number(6)
First_Name		Varchar(20)
Last_Name	Not null	Varchar(25)
Email	Not null	Varchar(25)
Phone_Number		Varchar(20)
Hire_date	Not null	Date
Job_id	Not null	Varchar(10)
Salary		Number(8,2)
Commission_pct		Number(2,2)
Manager_id		Number(6)
Department_id		Number(4)

(a) Find out the employee id, names, salaries of all the employees

```
SELECT employee_id, first_name, last_name,
Salary From employees;
```

(b) List out the employees who works under manager 100

```
SELECT employee_id, first_name, Last_name
From employees WHERE manager_id >=100;
```

(c) Find the names of the employees who have a salary greater than or equal to 4800

```
SELECT employee_id, first_name, Last_name
From employees WHERE Salary >=4800;
```

(d) List out the employees whose last name is 'AUSTIN'

```
SELECT employee_id, first_name, last_name FROM
employee WHERE last_name = 'AUSTIN';
```

(e) Find the names of the employees who works in departments 60,70 and 80

```
SELECT employee_id, first_name, last_name
From employees WHERE department_id IN
(60,70,80)
```

(f) Display the unique Manager_Id.

```
SELECT DISTINCT manager_id from employees;
```

Create an Emp table with the following fields: (EmpNo, EmpName, Job,Basic, DA, HRA,PF, GrossPay, NetPay) (Calculate DA as 30% of Basic and HRA as 40% of Basic)

```
create table Emp (
    Emp no int;
    Emp name varchar (50);
    Job varchar (25)
```

(a) Insert Five Records and calculate GrossPay and NetPay.

(b) Display the employees whose Basic is lowest in each department.

```
Select * from emp
where basic <= crosspay;
```

(c) If Net Pay is less than

```
select from emp
where netpay < grosspay;
```

a)

Employeed id	f-name	L-name	salary
1	rajesh	kumar	3000
2	kumar	kapoor	5000
3	devi	Priya	55000
4	jigal	Jarohi	25000

b)

Employee_id	first-name	last-name	e-mail
3	devi	Priya	devi@gmail.com
4	jigal	kapoor	jigal@gmail.com

c)

First_name	last-name
kumar	kapoor
Devi	Priya.

d)

Employee id	F-name	L-name	Phone-no
2	kumar	austin	83396120

e)

F-name	L-name
kumar	Austin
devi	Priya
jigal	Jaroor

f)

manager_id
200
300
100
100

Ex.No.: 3	WRITING BASIC SQL SELECT STATEMENTS
Date: 3/8/24	

Aim: The aim is to Implement & create the basic sql statements.

OBJECTIVES

After the completion of this exercise, the students will be able to do the following:

- List the capabilities of SQL SELECT Statement
- Execute a basic SELECT statement

Capabilities of SQL SELECT statement

A SELECT statement retrieves information from the database. Using a select statement, we can perform

- ✔ Projection: To choose the columns in a table
- ✔ Selection: To choose the rows in a table
- ✔ Joining: To bring together the data that is stored in different tables

Basic SELECT Statement

Syntax

```
SELECT *|DISTINCT Column_ name| alias
        FROM table_name;
```

NOTE:

DISTINCT—Suppress the duplicates.

Alias—gives selected columns different headings.

Example: 1

```
SELECT * FROM departments;
```

Example: 2

```
SELECT location_id, department_id FROM departments;
```

Writing SQL Statements

- SQL statements are not case sensitive
- SQL statements can be on one or more lines.

Using Literal Character String

- A literal is a character, a number, or a date included in the SELECT list.
- Date and character literal values must be enclosed within single quotation marks.

Example:

SELECT last_name||'is a'||job_id AS "EMPLOYEES JOB" FROM employees;

Eliminating Duplicate Rows

- Using DISTINCT keyword.

Example:

SELECT DISTINCT deparment_id FROM employees;

Displaying Table Structure

- Using DESC keyword.

Syntax

DESC table_name;

Example:

DESC employees;

Find the Solution for the following:

True OR False

1. The following statement executes successfully.

select employee_id, L_name, Salary*12 As Anacual_salary from employees;

Identify the Errors

SELECT employee_id, last_name
sal*12 ANNUAL SALARY
FROM employees;

Queries

2. Show the structure of departments the table. Select all the data from it.

Desc departments;
Select * from departments;

3. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first.

```
Select employee-id, l-name, job-id, hire-date
    from employee;
```

4. Provide an alias STARTDATE for the hire date.

```
SELECT hire-date AS startdate from
    employees;
```

5. Create a query to display unique job codes from the employee table.

```
Select Distinct job-id from employees;
```

6. Display the last name concatenated with the job ID , separated by a comma and space, and name the column EMPLOYEE and TITLE.

```
Select last-name || ', ' job-id As " Employee And
Title " from employees;
```

7. Create a query to display all the data from the employees table. Separate each column by a comma. Name the column THE_OUTPUT.

```
select employee_id || ',' || first_name ',' || Last
name ',' || email ',' phone_number || ',' || hire-date
" department id " As " the-output" From employees;
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	4
Total (15)	14
Faculty Signature	

1)

employee-id	Last-name	Annual-salary
101	Smith	72000
102	Doe	54000
103	Johnson	62400
104	Braun	36000

2)

Field	Type	Null	key	default	Extra
dep-id	int (11)	NO	PRI	NULL	-
dep-name	Var char(10)	No	-	NULL	-
manager-id	int (11)	YES	-	NULL	-
location-id	int (11)	YES	-	NULL	-

department id	department name	manager id
10	Administration	200 1700
20	marketing	201 1800
30	purchasing	202 1900
40	Human resource	203 2000
60	IT	204 2100

3)

Emp id	L-name	Job id	hire-date
101	Smith	IT-PROG	2020-01-15
102	Doe	HR-PREP	2019-02-18
103	Johnson	IT-PROG	2021-3-2d
104	Braun	AD-ASST	2018-04-22.

Ex.No.: 4	WORKING WITH CONSTRAINTS
Date: 7/8/24	

OBJECTIVE

After the completion of this exercise the students should be able to do the following

- Describe the constraints
- Create and maintain the constraints

What are Integrity constraints?

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies

The following types of integrity constraints are valid

a) **Domain Integrity**

✔ NOT NULL
✔ CHECK

b) **Entity Integrity**

✔ UNIQUE
✔ PRIMARY KEY

c) **Referential Integrity**

✔ FOREIGN KEY

Constraints can be created in either of two ways

1. At the same time as the table is created
2. After the table has been created.

Defining Constraints

Create table tablename (column_name1 data_ type constraints, column_name2 data_ type constraints …);

Example:

Create table employlees (employee_id number(6), first_name varchar2(20), ..job_id varchar2 (10), CONSTRAINT emp_emp_id_pk PRIMARY KEY (employlee_id));

(OR)
ALTER TABLE test 1 DROP(pk, fk, col1) CASCADE CONSTRAINTS;

VIEWING CONSTRAINTS

Query the USER_CONSTRAINTS table to view all the constraints definition and names.

Example:

```
SELECT constraint_name, constraint_type, search_condition FROM user_constraints
WHERE table_name='employees';
```

Viewing the columns associated with constraints

```
SELECT constraint_name, constraint_type, FROM user_cons_columns
WHERE table_name='employees';
```

Find the Solution for the following:

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column.The constraint should be named at creation. Name the constraint my_emp_id_pk.

```
ALTER table emp ADD constraints my_emp_id_
Primary key (id);
```

2. Create a PRIMAY KEY constraint to the DEPT table using the ID colum. The constraint should be named at creation. Name the constraint my_dept_id_pk.

```
ALTER table dept ADD constraint my_dept_id
Primary key (id);
```

3. Add a column DEPT_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to nonexistent deparment. Name the constraint my_emp_dept_id_fk.

```
ALTER Table emp ADD dept_id number;
ALTER Table emp ADD constraint my_emp_deptid
Foreign key (dept_id) reference dept(id);
```

4. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.

ALTER table emp ADD commision number(2,2);

ALTER table emp ADD constraint emp-commision.ck.

check (commision >0);

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	4
Total (15)	14
Faculty Signature	

1) Table Altered

2) Table Altered

3) Table Altered

4) Table Altered

Before Altering

Emp-Id	F-name	L-name	phoneno	Dep ID
101	John	Doe	555010	10
102	Jame	smith	5550101	20
103	Alice	Johson	5550103	30
104	Bob	Brown	5550104	40

After Altering:

Emp-ID	F-name	L-name	.phoneno	Dept-id
101	John	Joe	0.10	10
103	Alice	Johnson	0.15	30
104	Bob	Brown	0.12	40

Result:

Thus the given Query is executed Successfully.

Ex.No.: 5	
Date: 10/8/24	CREATING VIEWS

Aim:

To create views of employess table using Oracle syntax.

After the completion of this exercise, students will be able to do the following:

- Describe a view
- Create, alter the definition of, and drop a view
- Retrieve data through a view
- Insert, update, and delete data through a view
- Create and use an inline view

View

A view is a logical table based on a table or another view. A view contains no data but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables.

Advantages of Views

- To restrict data access
- To make complex queries easy
- To provide data independence
- To present different views of the same data

Classification of views

1. Simple view
2. Complex view

Feature	Simple	Complex
No. of tables	One	One or more
Contains functions	No	Yes
Contains groups of data	No	Yes
DML operations thr' view	Yes	Not always

Creating a view

Syntax

Use of WITH READ ONLY option.
Any attempt to perform a DML on any row in the view results in an oracle server error.

Try this code:

```
CREATE OR REPLACE VIEW empvu10(employee_number, employee_name,job_title)
AS SELECT employee_id, last_name, job_id
FROM employees
WHERE department_id=10
WITH READ ONLY;
```

Find the Solution for the following:

1. Create a view called EMPLOYEE_VU based on the employee numbers, employee names and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
Create view employee- As
Select employee id, lastname As employee
From employees.
```

2. Display the contents of the EMPLOYEES_VU view.

```
Select * From employee-vu;
```

3. Select the view name and text from the USER_VIEWS data dictionary views.

```
Select view-name, text
from user_views
where viewname = 'employee- vu';
```

4. Using your EMPLOYEES_VU view, enter a query to display all employees names and department.

```
Select employee, department id
from employee -vu;
```

5. Create a view named DEPT50 that contains the employee number, employee last names and department numbers for all employees in department 50.Label the view columns EMPNO, EMPLOYEE and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

```
Create view dept 50 As
Select emply_id As emp no, lastname As employee
Dept no
From employees, where dep-id = 50 with check option;
```

6. Display the structure and contents of the DEPT50 view.

```
Desc dept 50;
Select * from dept 50;
```

7. Attempt to reassign Matos to department 80.

```
update dept set dept no = 80 where emp = "mate
```

8. Create a view called SALARY_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the Employees, DEPARTMENTS and JOB_GRADE tables. Label the column Employee, Department, salary, and Grade respectively.

```
Create view salary-vu As
Select e_last name As employee id, departmen As
From employee
join department don e-department-id = d.depar-id
                                      Department
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	4
Total (15)	14
Faculty Signature	

Output

1. View Created

2.

Emp id	Emp	Dept id
101	smith	10
102	Johnson	20
103	williams	30

3.

View name	Text
Emp-vu	Select emp-id, last name As emp, dept-id from employees.

4.

Employee	Dept id
Smith	10
Johnson	20
Williams	30

5. View created.

6.

Name	Null?	Type
empno		Num (38)
employee		varchar (30)
Dept No		Num (58)

Ex.No.: 6	RESTRICTING AND SORTING DATA
Date: 13/8/24	

Aim :- To execute the following queries to restricting and sorting data.

After the completion of this exercise, the students will be able to do the following:

- Limit the rows retrieved by the queries
- Sort the rows retrieved by the queries
-

Limiting the Rows selected

- Using WHERE clause
- Alias cannot used in WHERE clause

Syntax

SELECT----------
FROM----------
WHERE condition;

Example:

SELECT employee_id,last_name, job_id, deparment_id FROM employees WHERE department_id=90;

Character strings and Dates

Character strings and date values are enclosed in single quotation marks.

Character values are case sensitive and date values are format sensitive.

Example:

SELECT emp'oyee_id,last_name, job_id, deparment_id FROM employees
WHERE last_name='WHALEN";

Comparison Conditions

All relational operators can be used. (=, >, >=, <, <= ,<>,!=)

Example:

SELECT last_name, salary

```
SELECT last_name, salary*12 annsal , job_id,department_id,hire_date
FROM employees
ORDER BY annsal;
```

Example:4

Sorting by Multiple columns

```
SELECT last_name, salary , job_id,department_id,hire_date
FROM employees
ORDER BY department_id, salary DESC;
```

Find the Solution for the following:

1. Create a query to display the last name and salary of employees earning more than 12000

```
select last-name salary from employees 6
where salary >12000;
```

2. Create a query to display the employee last name and department number for employee number 176.

```
select last-name, dept id from employee
where emp-id = 176;
```

3. Create a query to display the last name and salary of employees whose salary is not in the range of 5000 and 12000. (hints: not between)

```
select last-name , salary
from employees.
where salary not between 5000 & 12000;
```

4. Display the employee last name, job ID, and start date of employees hired between February 20,1998 and May 1,1998.order the query in ascending order by start date.(hints: between)

```
select last-name; job-id ; hire date from employee
where hire-date between '20-Feb-1998' AND
'1-may-1998' order by hire date;
```

1.

last Name	salary
King	24000.00
Kochas	17000.00
Dettan	170000

2.

last Name	Department - Id
Taylor	60

3.

last Name	salary
King	240000
Kochas	170000
Dettan	170000
Raphaely	11000.00

4.

last Name	Job id	Hire date
Grietz	AC - Account	21-MAR-1998
Taylor	SA - REP	24-APR-1998

5.

last Name	dept id
Abel	80
Ande	50
Grietz	20
Higgins	50

Ex.No.: 7	USING SET OPERATORS
Date: 17/8/24	

Aim:

To execute the Queries set operators.

Objectives

After the completion this exercise, the students should be able to do the following:

- Describe set operators
- Use a set operator to combine multiple queries into a single query
- Control the order of rows returned

The set operators combine the results of two or more component queries into one result.

Queries containing set operators are called *compound queries*.

Operator	Returns
UNION	All distinct 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 that are selected by the first SELECT statement and not selected in the second SELECT statement

The tables used in this lesson are:

- EMPLOYEES: Provides details regarding all current employees
- JOB_HISTORY: Records the details of the start date and end date of the former job, and the job identification number and department when an employee switches jobs

UNION Operator

Guidelines

- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are not ignored during duplicate checking.
- The IN operator has a higher precedence than the UNION operator.

Display the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired (that is, they changed jobs but have now gone back to
doing their original job).

```
SELECT employee_id, job_id FROM employees
INTERSECT
SELECT employee_id, job_id
FROM job_history;
```

Example

```
SELECT employee_id, job_id, department_id
FROM employees
INTERSECT
SELECT employee_id, job_id, department_id
FROM job_history;
```

MINUS Operator
Guidelines

- The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- All of the columns in the WHERE clause must be in the SELECT clause for the MINUS operator to work.

Example:

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id,job_id
FROM employees
MINUS
SELECT employee_id,job_id
FROM job_history;
```

Find the Solution for the following:

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST_CLERK. Use set operators to create this report.

```
Select dept_id from department MINUS
Select dept_id from employees where job_id = 'ST_CLERK';
```

2. The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use set operators to create this report.

```
Select country_id, country_name from countries
MINUS select country_id, country_name from
countries c, dept d where c.country_id
country_id;
```

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID using set operators.

```
Select job id , dept id
where dep-id IN (10, 50, 20)
Order by dept id;
```

4. Create a report that lists the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs but have now gone back to doing their original job).

```
Select emp-id , job-id from employees.
Intersect Select employee-id , job-id from job-history;
```

5. The HR department needs a report with the following specifications:

- Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department.

- Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them Write a compound query to accomplish this.

```
select last-name, dept-id from employees, union
Select dept-id , dept-name from department;
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	4
Total (15)	14
Faculty Signature	

1) Department table

dep-id	dept-name	country -id
10	Adim	US
20	HR	IN
50	sales	US
90	executive	
80	IT	
30		

2) Employees Table:

Emp -id	Job id	dept id	last name
100	SA - REP	50	king
101	ST-CLERK	20	kocchar
102	IT - PROG	10	Dehoan.
103	HR - REP		

3) Countries Table

country Id	county- name
US	united states
UK	United kingdom
IN	Indig

4) Job- History table

emp - Id	Job -id
101	SA - REP
103	HR - REP.

Result :-

Thus, the Queries are executed successfully.

WHERE emp.employee_id NOT IN (SELECT mgr.manager_id FROM employees mgr);

Notice that the null value as part of the results set of a subquery is not a problem if you use the IN operator. The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use the following SQL statement:

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id IN (SELECT mgr.manager_id FROM employees mgr);
```

Display all employees who do not have any subordinates:

```
SELECT last_name FROM employees
WHERE employee_id NOT IN (SELECT manager_id FROM employees WHERE manager_id
IS NOT NULL);
```

Find the Solution for the following:

1. The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).

```
Select last_name AS lastName entry year As (Hire Date)
From empl where dept id = (select dept id From empl
where last-name = 'Ztotkey') AND last Name <>'Zlotkey';
```

2. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary.

```
Select emp-id as Emp-ID, Last_Name AS 'Last Name',
Salary AS 'salary From Empl WHERE salary > (SELECT AVG
(salary) From emp ) ORDER by salary;
```

3. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a *u*.

```
Select emp-id , last-name from emp where emp-id
In ( Select dept id from emp WHERE (ast name
LIKE '%u%');
```

4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

```
SELECT last_Name AS 'Last Name', dept id As dept-no,
Job id AS 'Job-Id' From emp where dept-id From
emp 1 WHERE (loc id = 1700);
IN
select (dept-id)
```

5. Create a report for HR that displays the last name and salary of every employee who reports to King.

```
SELECT last_Name, salary From emp1 where
manager_name = 'King';
```

6. Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

```
Select dept id As Dept No', last-name AS 'Last name
job id As 'Job id' From emp1 WHERE emp id = (Select
dep id from emp1 WHERE dept-name = 'Executive');
```

7. Modify the query 3 to display the employee number, last name, and salary of all employees who earn more than the average salary and who work in a department with any employee whose last name contains a *u*.

```
Select emp1_id, last-name, Salary From emp1
WHERE Salary > (select Avg (salary) From emp1) AND
dept id IN (select distinct dept id From emp1
where last-name LIKE '%u%') order by
Salary Asc;
```

1)

Emp -id	last name	salary
101	Ramesh	10000
102	Suresh	120000
103	Devi	130000

2)

empl_id	last_name
106	Hasuto

3)

last-Name	Dept No	Job-Id
James	90	1003

4)

last-Name	Salary
Devi	130000

5)

Dept No	last-Name	Job Id
30	Hasuto	1005

Result:-

thus the Queries are executed successfully.

Ex.No.: 10	
Date: 27/8/24	AGGREGATING DATA USING GROUP FUNCTIONS

Aim :- To Create a database & execute the following Queries.

Objectives

After the completion of this exercise, the students be will be able to do the following:

- Identify the available group functions
- Describe the use of group functions
- Group data by using the GROUP BY clause
- Include or exclude grouped rows by using the HAVING clause

What Are Group Functions?

Group functions operate on sets of rows to give one result per group

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

Function	Description
AVG([DISTINCT|ALL] *n*)	Average value of *n*, ignoring null values
COUNT({*| [DISTINCT|ALL] *expr*})	Number of rows, where *expr* evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX([DISTINCT|ALL] *expr*)	Maximum value of *expr*, ignoring null values
MIN([DISTINCT|ALL] *expr*)	Minimum value of *expr*, ignoring null values
STDDEV([DISTINCT|ALL] *x*)	Standard deviation of *n*, ignoring null values
SUM([DISTINCT|ALL] *n*)	Sum values of *n*, ignoring null values
VARIANCE([DISTINCT|ALL] *x*)	Variance of *n*, ignoring null values

Group Functions: Syntax

SELECT [*column*,] *group_function*(*column*), ...
FROM *table*
[WHERE *condition*]

Group functions can be nested to a depth of two. The slide example displays the maximum average salary.

SELECT MAX(AVG(salary)) FROM employees GROUP BY department_id;

Summary

In this exercise, students should have learned how to:

- Use the group functions COUNT, MAX, MIN, and AVG
- Write queries that use the GROUP BY clause
- Write queries that use the HAVING clause

SELECT *column, group_function*
FROM *table*
[WHERE *condition*]
[GROUP BY *group_by_expression*]
[HAVING *group_condition*]
[ORDER BY *column*];

Find the Solution for the following:

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group.
True/False

2. Group functions include nulls in calculations.
True/False

3. The WHERE clause restricts rows prior to inclusion in a group calculation.
True/False

The HR department needs the following reports:

4. Find the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number

```
select around (max (salary),10)"max", round (min (salary,10)
"min"
, sum (salary) "sum", round (avg (salary),0 "Average
from empl;
```

5. Modify the above query to display the minimum, maximum, sum, and average salary for each job type.

```
select round (max (salary ,10)"max"), round (min ,sdom,0)
"min" , sum (salary) "sum", round ( avg (salary '0) "Avg"
from empl Group by Job Type;
```

6. Write a query to display the number of people with the same job. Generalize the query so that the user in the HR department is prompted for a job title.

```
Select   job_id , count (*) As total_employee from empl
Group by job_id ORDER BY total_ employee DESC;
```

7. Determine the number of managers without listing them. Label the column Number of Managers. *Hint: Use the MANAGER_ID column to determine the number of managers.*

```
select manager_Id count (*) As Numb_of_mang
from empl Group by manager_id;
```

8. Find the difference between the highest and lowest salaries. Label the column DIFFERENCE.

```
SELECT MAX (salary) - Min (salary) Diff
From empl;
```

9. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is $6,000 or less. Sort the output in descending order of salary.

```
Select manage_Id, MIN (salary) AS min_salary
from empl where manager_id is NOT_NULL Group
by manage,_id having min (salary) >6000.
```

10. Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

```
Select count (*) AS Total_employee,
count ( If (empty_year = 1995 ( 1 , NULL) As hired_19
from empl;
```

1.

Maximum	Minimum	Sum	Average
130000	12000	36200	90500

2.

Maximum	Minimum	Sum	Average
120000	10000	22000	11000
13 00000	13000	13000	13000
12000	12000	12000	12000

3.

Job id	total - employee
1001	2
1002	1
1003	1

4)

manager Id	no of manager
10	3
20	1

5. Difference

118000

6.

manager id	min salary
10	106000
20	12000

1)

Total emp	hired in 1995	1996	1997	1998
4	1	1	1	1

Ex.No.: 11	PL SQL PROGRAMS
Date: 3/10/12	

PROGRAMS

TO DISPLAY HELLO MESSAGE

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

PL/SQL procedure successfully completed.
```

TO INPUT A VALUE FROM THE USER AND DISPLAY IT

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

PL/SQL procedure successfully completed.
```

GREATEST OF TWO NUMBERS

```
SQL> set serveroutput on;

SQL> declare
  2  a number(7);
```

PROGRAM 1

Write a PL/SQL block to calculate the incentive of an employee whose ID is 110.

```
DECLARE
  Incentive NUMBER (8,2);
BEGIN
  Select Salary * 0.12 INTO Incentive
  From Employees
  where employee_id = 110;
  DBMS_OUTPUT.PUT_LINE ("Incentive =" || TO_CHAR
  END;                                (Incentive));
/
```

PROGRAM 2

Write a PL/SQL block to show an invalid case-insensitive reference to a quoted and without quoted user-defined identifier.

```
DECLARE
  "my variable" NUMBER = 100;
  my variable 2 NUMBER = 200;
BEGIN
  -valid reference to the Quoted Identifier
                    (case - sensitivity)
  DBMS_OUTPUT.PUT_LINE ('value of my variable': my variable)
  DBMS_OUTPUT.PUT_LINE ('Invalid Reference to my variable')
END;
/
```

PROGRAM 3

Write a PL/SQL block to adjust the salary of the employee whose ID 122.
Sample table: employees

```
DECLARE
   incentive   NUMBER (8,2);
BEGIN
   select  salary  * 0.12  INTO  incentive
   From  employees
   Where   employee_id = 110;
  DBMS - OUTPUT. PUT.LINE ('Incentive = " ||
   TO-CHAR (incentive);
  END;
/
```

PROGRAM 4

Write a PL/SQL block to create a procedure using the "IS [NOT] NULL Operator" and show AND operator returns TRUE if and only if both operands are TRUE.

```
DECLARE
  "my Variable" NUMBER = 100;
   my variable 2   NUMBER = 200;
BEGIN
  - Valid reference to the Quoted Identifier
    (case-sensitive must match exactly)
  DBMS - OUTPUT .PUT LINE ('value of my variable".
                                  my variable)
  DBMS - OUTPUT. PUT LINE ('value of my variable2.
                                  my variable");
  END;
```

PROGRAM 5

Write a PL/SQL block to describe the usage of LIKE operator including wildcard characters and escape character.

```
BEGIN
    UPDATE employees
     SET - salary = salary * 1.10
     WHERE   employee - id = 122;
     DBMS - OUTPOT . PUT- LINE ('salary adjusted for
                                employee with ID 122');
END;
/
```

PROGRAM 6

Write a PL/SQL program to arrange the number of two variable in such a way that the small number will store in num_small variable and large number will store in num_large variable.

```
CREATE OR REPLACE PROCEDURE Pri-bod (
     boo - name  VARCHAR 2,
     boo-val BOOLEAN
)Is
BEGIN
    IF bool- val IS NULL THEN
     DBMS - OUTPOT. PUTLINE (boo-name '' = NULL'');
ELSE IF     bool - val = TRUE THEN
    DBMS - OUTPOT . PUT - LINE (boo.name ||'' = TRUE');
ELSE
    DBMS - OUTPUT . PUT-LINE (boo - name ||'' =FALSE');
    END IF;
END; /
```

PROGRAM 7

Write a PL/SQL procedure to calculate the incentive on a target achieved and display the message either the record updated or not.

```
Declare
Procedure test1(
 sal- archieve number,
  target_qty Number,
  emp-id Number) is
 incentive Number:=0;
  updated varchar2(3) :='No';
BEGIN
 if sal-archieve >(target-qty+200) then
  Incentive = (sal- achieve - target-qty)14;
   updated emp
   where employee_id = empid;
   updated = 'yes';
   End if
DBMS-output.put-line
   ('Table updated?'||
     updated'||'
     'Incentive ='||
     incentive||!!
```

PROGRAM 8

Write a PL/SQL procedure to calculate incentive achieved according to the specific sale limit.

```
DECLARE
Procedure test1 (sal-achieve Number)is
 Incentive Number := 0;
Begin
   if sal-achieve > 4400 then
    Incentive = 1800;
else if sal-achieve > 32000 then
    incentive =800;
 else
    incentive; = 500;
end if
  Dbms-output.put-line;
  Dbms-output.put-line("sal-achieve ||' incentive:'||'incentive
end then;
Begin
test1 (68000);
```

Ex.No.: 12		WORKING WITH CURSOR, PROCEDURES AND FUNCTIONS
Date:	09/09/24	

AIM:

Create PL/SQL Blocks to perform the Item Transaction Operations using CURSOR, FUNCTION and PROCEDUERE.

ALGORITHM:

STEP-1: Start.

STEP-2: Create two tables Item Master and Item Trans.

itemmaster(itemid , itemname, stockonhand)

itemtrans(itemid ,itemname ,dateofpurchase ,quantity)

STEP-3: Create a PROCEDURE with id, name and quantity as parameters which make a call to the FUNCTION by passing id, name, dop, and quantity as parameters dop is set as sysdate.

STEP-4: Using FUNCTION fetch each record from the table Item Master using CURSOR inside a Loop statement,

If Item Master's ItemId is equal to the entered ID value then exit the loop otherwise fetch the next record.

```
loop
        fetch master into masterrec
        exit when master%notfound
        if masterrec.itemid=id then
    exit;
  end if;
end loop;
```

STEP-5: If Itemmaster's itemid = id then,

Add the Itemmaster's stockonhand with the given quantity and update the ItemMaster table and insert the Item information into the ItemTrans table.

STEP-6: Else, if the inputed item is not present in the ItemMaster table then insert the

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

```
Create or replace Function factorial
                (p_number in number)
  Return number is
     v_ return number := 1;
Begin
     if p_number < 0 then
      return Null;
   End if;
For    i in p_number loop
     v_ result = v_ result * i;
End   loop;
    Return   v_ result;
   End;
/
```

Program 2

Write a PL/SQL program using Procedures IN,INOUT,OUT parameters to retrieve the corresponding book information in library

```
Create or replace procedure set_booh_info (
p_book_id (N Number)
b_tiffle_OUT varchar2;
p_author OUT varchar2;
b_pub_year OUT Number,
p_status IN OUT varchar2) is
Begin
  p_tiffle = NULL;
  p_author = NULL;
  p_pub_year = NULL;
  Select tiffle, author, publication_year
  into p_tiffle, p_author, b_year
  From library
  where booh_id = p_booh_id;
  if p_tiffle is Not Null then
     p_status = 'Booh found';
  else
     p_status = 'Booh Not Found';
  end if
exception
  when No_data_Found then
     p_status = 'Booh not Found';
     p_tiffle := NULL;
     p_author := NULL;
     p_pubyear = NULL;
end;
/
```

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

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

Program 1

Write a code in PL/SQL to develop a trigger that enforces referential integrity by preventing the deletion of a parent record if child records exist.

```
Create or replace Trigger prevent - Parent - deletion
  Before delete on departments.
  For each row
  Declare
      v-count Number;
Begin
    Select count (*) into v-count From employees where
    if v-count >0 then                    emp-id = old. dep-id;
    Raise- Application -Error (-200 1/1 'cannot delete dep
                                  with associate emp');
    end;/
```

Program 2

Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found.

```
Create Table Products (
    Product_id Number Primary key
      Product-name Varchar (50);
Create or replace Trigger prevent-duplicates
  Before insert Or Products
  For each row
  Declare
    v- count Number;
Begin
  Select count (*) into v- count From products
                where prod- name = New Prod - name;
  End;
```

Program 3

Write a code in PL/SQL to create a trigger that restricts the insertion of new rows if the total of a column's values exceeds a certain threshold.

```
Create Table Orders (
    Order_id num primary key;
    customer_id number,
    Order_amount Number);
Create Or Replace Trigger check_order
Before insert On orders_amount
    For each row
Declare
    total_amount Number;
    Max_threshold Number=1000;
Begin
    select NULL
    from orders
    where cust_id = New.
        If total_amt
    new order
    max_threshold then
    raise Application
    Error.
```

Program 4

Write a code in PL/SQL to design a trigger that captures changes made to specific columns and logs them in an audit table.

```
Create or replace Trigger salary_change_audit
    After update on employees
    For each row
    when (New salary <> old salary)
Declare
    v_audi_id Number;
Begin
    select seq_salary_audit Next val into v_audit_id)
Insert into salary_audit (audit_id, emp_id,
                          old_sal, new_sal)
value (v_audit_id, old emp_id, :old sal:,
                          sal, sysdt);
```

Program 5

Write a code in PL/SQL to implement a trigger that records user activity (inserts, updates, deletes) in an audit log for a given set of tables.

```
Create table employee(
  emp-id Number primary key,
emp-name varchar2 (100),
  emp-salary Number );
Create table Audit log(
  log-id Number primary key,
  take-name Varchar (100),
  activity-type varchar(20),
   user-id varchar (50);
```

```
Begin
if Inserting then
v-activity-type = 'Insert';
else if updating then
v-activity-type = 'del'
end if;
end;
```

Program 7

Write a code in PL/SQL to implement a trigger that automatically calculates and updates a running total column for a table whenever new rows are inserted.

```
Create Table Sales(
  Sales-id Number primary key,
  Sales-date Date,
   amount Number,
  running-total number;
Create or replace Trigger update-running-Total
  Before insert on Sales
  for each row
  Begin
    if :New running total is Null then
  end if;
  End;
  /
```

Program 8

Write a code in PL/SQL to create a trigger that validates the availability of items before allowing an order to be placed, considering stock levels and pending orders.

```
Create table Products(
  Product_id Number primary key,
  Product_name varchar2(100),
  Stock_Quantity Number);
Create table orders(
  Order_id Number primary key,
  Product_id Number,
  Order_Quantity Number);
Create or replace trigger validate
Before insert Order
For each row
Declare
Begin
Select stock_Quantity into
  v_current_stock
From products
where product_id:=new_
                 prod_id;
If v_curr_stock<v_pend
  Order;
  New_orde Q<0
then
  (-20001,'Insufficient
          stock for orde');
end if;
end;
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	5
Program/Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

date: Date()

})

EXERCISE 18

Structure of 'restaurants' collection:


```
{
 "address": {
  "building": "1007",
  "coord": [ -73.856077, 40.848447 ],
  "street": "Morris Park Ave",
  "zipcode": "10462"
 },
 "borough": "Bronx",
 "cuisine": "Bakery",

 "grades": [
  { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
  { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
  { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
  { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
  { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
 ],
 "name": "Morris Park Bake Shop",
 "restaurant_id": "30075445"

}
```


1. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees' or restaurant's name begins with letter 'Wil'.

```
db.restaurant.find({
$or [{cuisine :{ $in: ["American", "chinese"]}},
{name: { $regex: /^Wil/, $options: "i"}}]}{restaurant_id:1, name:1, borough:1});
```

2. Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among many of survey dates.

```
db.restaurant.find({
"grades": {$elemMatch: {"grade": "A", "score":11
"date": ISODate("2014-08-11T00:00:00Z")}}});
(restaurant_id:1, name:1, grade:1});
```

3. Write a MongoDB query to find the restaurant Id, name and grades for those restaurants where the 2nd element of grades array contains a grade of "A" and score 9 on an ISODate "2014-08-11T00:00:00Z".

```
db.restaurant.find({"grades.1.grade": "A",
"grades.1.score": 9, "grades.1.date": ISODate("2014-08-11")},
{restaurant_id:1, name:1, grades:1});
```

4. Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of coord array contains a value which is more than 42 and upto 52..

```
db.restaurant.find({
"address.coord.1": {$gt:42, $lte:52}},
{restaurant_id:1, name:1, address:1, "address.coord":1});
```

5. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.

```
db.restaurants.find().sort({name:1});
```

6. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.

```
db.restaurants.find().sort({name:-1});
```

7. Write a MongoDB query to arranged the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

```
db.restaurants.find().sort({cuisine:1, brought:-1});
```

8. Write a MongoDB query to know whether all the addresses contains the street or not.

```
db.restaurants.find({"address.street":{$exists:true}});
```

9. Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double.

```
db.restaurants.find ({"address.coord": { $type: "double"
                                        }})
```

10. Write a MongoDB query which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing the score by 7.

```
db.restaurants.find ({
  "grades" : { $elemmatch: {"score": { $mod: [7,0]}}},
{ restaurant_id : 1, name:1, grades:1});
```

11. Write a MongoDB query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contains 'mon' as three letters somewhere in its name.

```
db.restaurants.find ({
    name: { $regex : "mom", $option : "i"}},
{ name:1, brough:1," address.coord": 1, cuisine:1});
```

12. Write a MongoDB query to find the restaurant name, borough, longitude and latitude and cuisine for those restaurants which contain 'Mad' as first three letters of its name.

```
db.restaurants.find ({
    name:{ $regex : "mad", { option : "i"}},
{ name:1, borough:1, "address.coord": 1, cuisine :1});
```

13. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5.

```
db.restaurants.find ({
"grades.{ elemmateh :{ "score: {$lt:5}}});
```

14. Write a MongoDL query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan.

```
db.restaurants.find ({
"grades": { $ elemmatch: {"score": { $ lt:5}}}
   boroug : "manhattan"})
});
```

15. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn.

```
db.restaurants.find({
"grades": { $elemmatch: { "score": { $lt: 5 } } },
borough: { $in: ["manhattan", "Brooklyn"] } });
```

16. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.

```
db.restaurants.find({
"grades": { $elemmatch: { "score": { $lt: 5 } },
borough: { $in: ["manhattan", "Brookyln"] },
cuisine: { $ne: "American" } });
```

17. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.

```
db.restaurants.find({
"grades": { $elemmatch: { "scores": { $lt: 5 } } },
borough: { $in: ["manhattan", "Brookyln"] },
```

18. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6.

```
db.restaurants.find({
grades: { $all: [{ $elemmatch: { score: 2 } },
{ $elemmatch: { score: 6 } }] });
```

1) CREATE SEQUENCE Dept-ID
START WITH 200 → INCREMENT BY 10
MAX VALUE 1000
No CACHE;
No CYCLE

2) SELECT Sequence-name, Max-value, Increment-by last number From User-sequences;

3) Insert into Dept (ID, Dept-name) values
('Dept-ID', 'Next val', 'education');

Insert into Dept (ID, Dept-name) VALUES
(Dept-ID, Next val, "Administration");

commit;

Select * from Dept where Dept-name ("education Administration")

4) CREATE INDEX dept-id-emp ON Emp (Dept-id);

5) SELECT INDEX-Name, Uniqueness from user-Index
WHERE Table-name = 'EMP';

1. To allow user to log on to an Oracle Server, they must be granted the CREATE SESSION privilage. This is a system privilage, not an object privilage, an it allows the user to establish a session & connect to the oracle database server.

"Grant create session to username;"

2. To allow a user to create tables in an Oracle database, they must be granted the Create table privilage This is a system privilage.

"Grant create table to username;";

3. If you create a table in an oracle database, only you, as the table owner, or a user with the Admin options or Grant option on the relevant privilage.

"Grant select on your-table to username with Grant option";

4) As I am a DBA, to simplify the process of granting the same set of privilages to many users.

6. Grant another user access to your Department Table. Assume your username is user A & you want to grant user B access to your Dep table.

7. To Query all the rows from your Dep table, You would use a simple select statement:

"Select * From departments";

8. Insert into departments (dep_id, dep_name) values (500, 'Education');

9. "Select table_name, tablespace_name, num_rows, blocks From user_tables;"

10. "Revoke select ON department from team2;"

11. Team 1: Remove the education dep

"Delete From dep (dep num 500);"

Team 2: Remove the Human Resources dep

"Delete from dep where dep_id = 510";