Ex.No.: 1	CREATION OF BASE TABLE AND DML OPERATIONS
Date: 26/7	

AIM:

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>;

STEP-8: The truncate command deletes all rows from the table. Only the structure of the table remains.
Syntax:

TRUNCATE TABLE <table name>;

STEP-9: Alter the existing table using ALTER statement.
Syntax:

Add Column:

ALTER TABLE <table name> ADD (column data type [DEFAULTexpr][,column data type]);

Modify Column:

ALTER TABLE <table name> MODIFY (column data type [DEFAULT expr], [,column data type]);

Drop Column:

ALTER TABLE <table name> DROP COLUMN <column name>;

STEP-10: To drop the entire table using DROP statement.
Syntax:

DROP TABLE <table name>;

STEP-11: Exit.

1. Create MY_EMPLOYEE table with the following structure

create table MY_EMPLOYEE(emp_no number(4, last_name varchar(29), First_name varchar(25), User id varchar(25), salary(9,2));

NAME	NULL?	TYPE
ID	Not null	Number(4)
Last_name		Varchar(25)
First_name		Varchar(25)
Userid		Varchar(25)
Salary		Number(9,2)

2. Add the first and second rows data to MY_EMPLOYEE table from the following sample data.

ID	Last_name	First_name	Userid	salary
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	Cnewman	750
5	Ropebur	Audrey	aropebur	1550

```
Insert into MY_employee values (1,'Patel','Ralph','rpatel',895);
Insert into MY_employee values (2,'dancs','betty','bdancs',860);
```

3. Display the table with values.

```
Select * from MY_EMPLOYEE;
```

4. Populate the next two rows of data from the sample data. Concatenate the first letter of the first_name with the first seven characters of the last_name to produce Userid.

```
insert into MY_employee values (3,'biri','ben',
concat(substr("First_name",1,1),substr('last_name',1,7));
```

5. Delete Betty dancs from MY _EMPLOYEE table.

```
Delete MY_EMPLOYEE where (first name = 'betty' and
last_name = 'dancs');
```

6. Empty the fourth row of the emp table.

delete MY_EMPLOYEE where (ID=4);

7. Make the data additions permanent.

commit;

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

update table 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 table MY_EMPLOYEE set (salary=1000) where salary < 900.

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

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

Create the following tables with the given structure.

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 emp_id, First_name, last_name, salary from employee;

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

Select * From employee where mager_ID = 100;

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

Select First_name, last_name from employee where salary >= 4800;

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

```
Select * From employee where last_name like 'austin';
```

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

```
Select First_name, last_name from employee where dept_id
in (60,70,80);
```

(f) Display the unique Manager_Id.

```
Select distinct manger_id from employee;
```

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(25), job
varchar(20), basic int, da int, hra int, PI varchar(25),
gross pay int, net pay int);
```

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

```
Insert into emp 101, 'Moun', manage, 15000, 0,0,0,0),
(102, 'Rick', 'developer', 7500, 0,0,0,0).
```

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

```
Select * from emp
where basic in (select min(basic) as basic, from emp
group by job);
```

(c) If Net Pay is less than 100 000

```
Select * from emp where netpay < 150000;
```

DEPARTMENT TABLE

NAME	NULL?	TYPE
Dept_id	Not null	Number(6)
Dept_name	Not null	Varchar(20)
Manager_id		Number(6)
Location_id		Number(4)

JOB_GRADE TABLE

NAME	NULL?	TYPE
Grade_level		Varchar(2)
Lowest_sal		Number
Highest_sal		Number

LOCATION TABLE

NAME	NULL?	TYPE
Location_id	Not null	Number(4)
St_addr		Varchar(40)
Postal_code		Varchar(12)
City	Not null	Varchar(30)
State_province		Varchar(25)
Country_id		Char(2)

1. Create the DEPT table based on the DEPARTMENT following the table instance chart below. Confirm that the table is created.

Column name	ID	NAME
Key Type		
Nulls/Unique		
FK table		
FK column		
Data Type	Number	Varchar2
Length	7	25

Create table dept (id_ number(7), name, varchar(25)).

2. Create the EMP table based on the following instance chart. Confirm that the table is created.

Column name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK table				
FK column				
Data Type	Number	Varchar2	Varchar2	Number
Length	7	25	25	7

create table emp (id number (7), last_name varchar (25), first_name varchar (25), dept_id number (7));

3 Modify the EMP table to allow for longer employee last names. Confirm the modification.(Hint: Increase the size to 50)

alter table emp1 modify last_name varchar2(25);

4 Create the EMPLOYEES2 table based on the structure of EMPLOYEES table. Include Only the Employee_id, First_name, Last_name, Salary and Dept_id coloumns. Name the columns Id, First_name, Last_name, salary and Dept_id respectively.

create table employee2 (id number(2), first_name varchar(25), last_name varchar(25), salary number(8,2), dept_id number (4));

5 Drop the EMP table.

drop table emp1;

6 Rename the EMPLOYEES2 table as EMP.

alter table employee2 rename to emp1;

7 Add a comment on DEPT and EMP tables. Confirm the modification by describing the table.

comment on table dept is 'information of employee dept';
comment on table emp1 is 'information of employee details'.

8 Drop the First_name column from the EMP table and confirm it.

Alter table emp1 drop column first_name;

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

Ex.No.: 3		WRITING BASIC SQL SELECT STATEMENTS
Date:	2/8	

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.

Identify the Errors

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

select employee_id, last_name, sal*12 as Annual_Salary from employees;

Queries

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

desc dept;
insert into dept (id, name) select 9, first_name from employee;
select * From dept;

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 emp_id, last_name, job_id, hire_date from employees;

4. Provide an alias STARTDATE for the hire date.

Alter table employees rename column hire_date to start_date;

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

Select last_name || ', ' || job_id as employee and 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 distinct job_id 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 || ',' || job_id as output from employees.

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

Ex.No.: 4		WORKING WITH CONSTRAINTS
Date:	6/8	

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 constraint my_id_PK
Primary key (employee_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_PK
Primary key (Dept_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 constraint my_emp_dept_id_PK
Foreign key (dept_id) reference dept 2(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 commission number(2,2)
Alter table emp add constraint check_ commission
check (commission >0);

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

Ex.No.: 5		CREATING VIEWS
Date:	13/8	

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 emp_id, first_name ||
last_name as employee, dept_id from employees;
```

2. Display the contents of the EMPLOYEES_VU view.

```
Select * from employee_vie;
```

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

```
Select view_name, text from user_views
where user_name = 'Employee_VU';
```

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

```
Select employee, dept_id from employee.vie;
```

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 dept50 as select emp_id as emp no,
last name as employee, department_number as dept no from
employees where department_number = 50 with check option;
```

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

```
Describe Dept 50;
Select * From dept 50;
```

7. Attempt to reassign Matos to department 80.

```
Update Dept 50 set dept no = 80 where employee = 'matos';
```

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 no,
d.department_name as department
e.salary as salary, j.grade as grade;
From employees e join department d on e.department_id =
d.department_id
Join job grade;
on e.salary
between j.lowest_
salary and j.
highest salary;
```

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

Ex.No.: 6		RESTRICTING AND SORTING DATA
Date:	2018	

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 employee_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
where salary >12000;
```

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

```
Select last_name, department_id
From employees
where employee_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 and 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 employees
where hire_date between '20-Feb-1998' and '01 -
order by hire_date;
```

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.(hints: in, orderby)

```
Selat last_name, department_id
From employees
where department_id in (20,50)
order by last_name;
```

6. Display the last name and salary of all employees who earn between 5000 and 12000 and are in departments 20 and 50 in alphabetical order by name. Label the columns EMPLOYEE, MONTHLY SALARY respectively.(hints: between, in)

```
Select last_name "Employee", Salary "monthly salary"
From employees
where salary between 5000 and 12000
And department_id in (20,50);
```

7. Display the last name and hire date of every employee who was hired in 1994.(hints: like)

```
Select last_name, hire_date
  From employees
where hire_date like '%94';
```

8. Display the last name and job title of all employees who do not have a manager.(hints: is null)

```
Select last_name, job_id
From employees
where manager_id is null;
```

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.(hints: is not nul,orderby)

```
Selat last_name, salary, commission_pet
From employees
where commission_pet is not NULL
Order by salary Desc, commission_pet Desc,
```

10. Display the last name of all employees where the third letter of the name is *a*.(hints:like)

```
Selat last_name
From employees
where last_name like '_a%';
```

11. Display the last name of all employees who have an a and an *e* in their last name.(hints: like)

```
Selat last_name
From employees
where last_name like '%a%' and
    last_name like '%e%';
```

12. Display the last name and job and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to 2500 ,3500 or 7000.(hints:in,not in)

```
Selat last_name, job_id, salary
From employees
where job_id in ('SA_REP',
'ST_CLERK') And salary not
in (2500, 3500, 7000);
```

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

Ex.No.: 7		
Date:	27/8	USING 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.

1) Select department_id
From employees
minus
Select department_id
From employees
where upper(job_id) = upper('ST_CLERK')
order by 1;

2) Select country_id, country_name
From countries
minus
Select country_id, country_name
From countries c
join locations l
using (country_id)
join departments d
using (location_id)
where department_id is not null;

3) Selat distinct job_id, department_id
From employees
where department_id = 10
union all
Select distinct job_id, department_id
From employees
where department_id = 50
union all
Selat distinct job_id, department_id
From employees
where department_id = 20;

4) Selat employee_id, job_id
From employees
Intersect
Select employee_id, job_id
From job_history
Order by 1;

5) Selat last_name, department_id, TO_char('null')
Fromemployees
Union
Select to_char('null'), department_id, department_name
From departments
Order by 1;

Ex.No.: 8		
Date:	3/9	WORKING WITH MULTIPLE TABLES

Objective

After the completion of this exercise, the students will be able to do the following:
• Write SELECT statements to access data from more than one table using equality and nonequality joins
• View data that generally does not meet a join condition by using outer joins
• Join a table to itself by using a self join
Sometimes you need to use data from more than one table.

Cartesian Products
• A Cartesian product is formed when:
– A join condition is omitted
– A join condition is invalid
– All rows in the first table are joined to all rows in the second table
• To avoid a Cartesian product, always include a valid join condition in a WHERE clause.
A Cartesian product tends to generate a large number of rows, and the result is rarely useful. You should always include a valid join condition in a WHERE clause, unless you have a specific need to combine all rows from all tables.
Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

Example:
To displays employee last name and department name from the EMPLOYEES and DEPARTMENTS tables.

```
SELECT last_name, department_name dept_name
FROM   employees, departments;
```

Types of Joins

- Equijoin
- Non-equijoin
- Outer join
- Self join
- Cross joins
- Natural joins
- Using clause
- Full or two sided outer joins
- Arbitrary join conditions for outer joins

Joining Tables Using Oracle Syntax

```
SELECT table1.column, table2.column
```

This query was completed in earlier releases as follows:

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  d.department_id = e.department_id (+);
```

FULL OUTER JOIN
Example:

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
FULL OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It alslso retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

Find the Solution for the following:

1. Write a query to display the last name, department number, and department name for all employees.

```
Select e.last_name, e.department_id, d.department_name
From employees e. department d
where e.department_id = d.department_id;
```

2. Create a unique listing of all jobs that are in department 80. Include the location of the department in the output.

```
Select distinct job_id, location_id
From employees, departments
where employees, department_id = departments.department_id
And employees.department_id = 80;
```

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission

```
select e.last_name, d.department_name, d.location_id, l.city
From employees e, departments d, locations l
where e.department_id = d.department.id
And
d.location_id = l.location_id
and e.commission_pct is not null;
```

2. Display the employee last name and department name for all employees who have an a(lowercase) in their last names. P

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, Respectively

```
Select w.last_name "Employee", w.employee_id "Emp#",
    m.last_name "manager", m.employee_id "mgr#"
From employees w Join employees m
on (w.manager_id = m.employee_id);
```

7. Modify lab4_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

```
Select w.last_name "Employee", w.employee_id "Emp#",
    m.last_name "manager", m.employee_id "MGR #"
From employees w
Left Outer join employees m
on (w.manager_id = m.employee_id);
```

8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label

```
Select e.department_id department, e.last_name employee,
    c.last_name colleague
From employees e Join employees c
On (e.department_id = c.department_id)
where e.employee_id <> c.employee_id
order by e.department_id, e.last_name, c.last_name;
```

9. Show the structure of the JOB_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees

```
desc job_grades
Select e.last_name, e.job_id, d.department_name, e.salary, j.grade_level
From employees e, department d, job_grades j
where e.department_id = d.d.department_id
And e.salary Between j.lowest_sal and j.highest_sal
```

10. Create a query to display the name and hire date of any employee hired after employee Davies.

```
Select e.last_name, e.hire_date
From employees e join employees davies
on (davies.last_name = 'davies')
where davies.hire_date < e.hire_date;
```

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively.

```
Select w.last_name, w.hire_date, m.last_name, m.hire_date
From employees w join employees m
on (w.manager_id = m.employee_id)
where w.hire_date < m.hire_date;
```

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

Ex.No.: 9		SUB QUERIES
Date:	10/9	

Objectives

After completing this lesson, you should be able to do the following:
- Define subqueries
- Describe the types of problems that subqueries can solve
- List the types of subqueries
- Write single-row and multiple-row subqueries

Using a Subquery to Solve a Problem
Who has a salary greater than Abel's?

Main query:
Which employees have salaries greater than Abel's salary?

Subquery:
What is Abel's salary?

Subquery Syntax

SELECT *select_list* FROM *table* WHERE *expr operator* (SELECT *select_list* FROM *table*);

- The subquery (inner query) executes once before the main query (outer query).

- The result of the subquery is used by the main query.

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses, including the following:

- WHERE clause
- HAVING clause
- FROM clause

In the syntax:
operator includes a comparison condition such as >, =, or IN

Note: Comparison conditions fall into two classes: single-row operators

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, hire_date from employees
where department_id = (Select department_id from employees
where last_name like ('&name') and last_name <>'&name');
```

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 employee_id, last_name, salary
From employees
where salary > (Select avg(salary) from employees)
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 employee_id, last_name
From employees
where department_id in (select department_id from
employees where last_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.

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 employees
where manager_id in (Select
employee_id from employees where last_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 department_id, last_name, job_id from employee
where department_id in (Select department_id from
departments where department_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 employee_id, last_name, salary from employee
where salary > (Select avg(salary) from employees
and department_id in (Select department_id from
employees where last_name like '%u%');
```

Ex.No.: 10		AGGREGATING DATA USING GROUP FUNCTIONS
Date:	20/9	

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 True

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

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

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 max(salary) as maximum, Avg(salary) as average,
Sum(Salary) as total, min(Salary) as minimum from employees;
```

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

```
Select current_job_id, max(salary) as maximum, Round(Avg(salary))
as average, Sum(salary) as total, min(salary) as minimum
From employees
Group by current_job_id;
```

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 current_job_id, count(current_job_id) as jobs
From employees
Group by current_job_id;
```

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 count(current_job_id) as "Number of managers"
from employees
where current_job_id like '%MGR%';
```

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

```
Select max(salary) - min(salary) as difference
From employees;
```

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 reports_to, min(salary)
From employees;
where salary <= 6000 and reports_to is not null
Group by reports_to
Order by min(salary) desc;
```

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,
Sum(if(hire_date like '1995%', 1, 0)) as '1995',
Sum(if(hire_date like '1997%', 1, 0)) as '1997',
Sum(if(hire_date like '1998%', 1, 0)) as '1998',
From employees;
```

11. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

```
Select current_job_id AS 'JOB'
Sum(if(department_id=20, salary, 0)) as 'Dept 20',
Sum(if(department_id=50, salary, 0)) as 'Dept 50',
Sum(if(department_id=80, salary, 0)) as 'Dept 80',
Sum(salary) as 'Total', From employees, Group by current_job_id;
```

12. Write a query to display each department's name, location, number of employees, and the average salary for all the employees in that department. Label the column name-Location, Number of people, and salary respectively. Round the average salary to two decimal places.

```
Select d.department_name as "Department name",
d.location as "location",
Count(e.employee_id) as "number of people"
Round(Avg(e.salary), 2) AS "Salary" From department d
left Join employees e on d.department_id = e.department_id
Group by d.department_name, d.location;
```

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

Ex.No.: 11		PL SQL PROGRAMS
Date:	2/9	

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
emp_id employees.emp_id%.Type:=110;
emp_id employees.name%.Type;
emp_salary employees.Salary%.Type;
 incentive number(7,2);
BEGIN
Select name, salary
Into emp_name, emp_salary
From employees
where emp_id=110;
incentive := emp_salary *0.10;
DBMS_output.PUT_line('Employee name:'||emp_name);
DBMS_output.Put_line('Incentive (10%):'|| incentive);
```

PROGRAM 2

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

```
Set Serveroutput ON;
Declare
   employee varchar (50); :='John Doe';
   "Employee" varchar (50); :='Jane Doe';
   Begin
     DBMS_output.Put_line('case_Insitive (employee name:'||
DBMS_output.Put_line('case_sensitive ("employee name"):'||
Exception
     DBMS_output.Put_line('Error:'||SQLERRM);
End;
```

PROGRAM 3

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

```
Set Serveroutput on;
Begin
   update employees
   Set salary = salary + (salary * 0.10)
   where emp_id = 122
   Returning salary INTO :new_salary
   DBMS_output.Put_line('New Salary :' || :new_salary);
Exception
   when NO_data_Found then
      DBMS_output.put_line('Employee with ID 122 not found.');
when Others then
   DBMS_output.Put_line('Error :' || SQLERRM);
   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.

```
Set Serveroutput on;
Begin
   If ('Hello' is not null and null IS NOT NULL) Then
      DBMS_output.put_line('Both are not NULL');
   Else
      DBMS_output.put_line('Atleast one is NULL');
      END IF;
END;
/
```

OP: Atleast one is NULL

PROGRAM 5

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

```
Set Serveroutput On;
Begin
   If 'Hello world' LIKE 'H%w%' Then
   DBMS_Output_Line ('Pattern 1 matched.');
   End If;
  If 'Hello 123' like 'Hello_123' Then
    DBMS_Output.Put_line ('Pattern 2 matched.');
    End If;
  If 'SO% discount' like 'SO \%%' Escape '\' Then
    DBMS_Output.Put_Line ('Pattern 3 matched with escape');
```

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.

```
Set serveroutput on;
Declare
    num1 number := 10;
    num2 number := 20;
     num_small number := least (num1, num2)
     num_large number := Greatest (num1, num2);
Begin
   DBMS_Output.Put_Line ('Small :' || num_small ||
     large :' ||
   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.

```
Set serveroutput on;
create or replace Procedure calc_incentive (emp_id in number) is
Begin
update employees set incentive = target_achieved * 0.10 where emp_id
DBMS_output.put_line ('Record' || case when SQL%ROWCOUNT > 0 then
'updated.
!= 'Record not updated.' END);
END;
```

PROGRAM 8
Write a PL/SQL procedure to calculate incentive achieved according to the specific sale limit.

```
Set Serveroutput ON;
create or replace Procedure procedure calc_incentive (emp_id in number)
Sales_limit_number = 1000;
incentive
Begin
  Select case when total_sales >= sales_limit then total_sales * 0.10
update employees set incentive = incentive_amount where emp_id = emp_id
DBMS_output.put_line ('Incentive for ID' || emp_id || emp_id || ':' ||);
Exception
  when NO_Data_Found then DBMS_output.put_line('Employee not
Found!.');
End;
```

PROGRAM 9

Write a PL/SQL program to count number of employees in department 50 and check whether this department have any vacancies or not. There are 45 vacancies in this department.

```
Set server output on;
Declare
  emp_count number;
Begin
  Select count(*) Into emp_count From employees where dept_id = 50;
DBMS_output.put_line('employees in Dept 50:'||emp_count);
DBMS_output.put_line(IIF(emp_count < 45, 'Vacancies available'));
END;
/
```

PROGRAM 10

Write a PL/SQL program to count number of employees in a specific department and check whether this department have any vacancies or not. If any vacancies, how many vacancies are in that department.

```
Set server output on;
Declare
  emp_count number;
  Vacancies number := 45;
Begin
  Select count(*) INTO emp_count From employees where departme
DBMS_output.put_line('Employees in Dept 50:' ||emp_count|| 'Vacancies
(Vacancies_emp_count));
END;
```

PROGRAM 11

Write a PL/SQL program to display the employee IDs, names, job titles, hire dates, and salaries of all employees.

```
Set serveroutput on;
Begin
For rec IN (Select employee_id, name, job_title, hire_date, salary
From
DBMS_output.put_line('ID:' || rec.employee_id ||
', Name:' || rec.name ||
', Job title:' || rec.job.title ||
', Hire Date:' || rec.hire_date ||
', Salary:' || rec.salary);
END loop;
END;
```

PROGRAM 12

Write a PL/SQL program to display the employee IDs, names, and department names of all employees.

```
Set serveroutput on;
Begin
For rec IN (Select e.employee_id, e.name, d.department_name
From employees e
Join department d on e.department_id = d.departmen
DBMS_output.put_line('ID:' || rec.employees_id ||
', name:' || rec.name ||
', Department:' || rec.department_name);
END loop;
END;
```

PROGRAM 13
Write a PL/SQL program to display the job IDs, titles, and minimum salaries of all jobs.

```
Set Serveroutput ON;
Begin
  For rec IN (Select job_id, job_title, min_salary From jobs)
DBMS_output.put_line('Job ID: '||rec.Job_id)
                    ||', Title: '||rec.job_title||
                    ||', min Salary: '||rec.min_salary);
END loop;
END;
```

PROGRAM 14
Write a PL/SQL program to display the employee IDs, names, and job history start dates of all employees.

```
set Serveroutput on;
Begin
  For rec IN (Select job_id, job_title, min_salary From jobs)loop
DBMS_output.put_line('ID '||rec.employee_id||
                    ', Name: '||rec.name||
                    ', Job start Date: '||rec.end_date);
END loop;
END;
```

PROGRAM 15

Write a PL/SQL program to display the employee IDs, names, and job history end dates of all employees.

```
Set serveroutput on;
Begin
    For rec In (Select e.employee_id, e.name, j.end_date
    From employees e
    Join job_history j On e.employee_id = j.employee_id)
    DBMS_Output.put_line('ID: ' || rec.employee_id
        || ', Name: ' || rec.name
        || ', Job End Date: ' || rec.end_date);
    END LOOP;
END;
```

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

Ex.No.: 12		WORKING WITH CURSOR, PROCEDURES AND FUNCTIONS
Date:	27/9	

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 ( n IN number ) Return number;
 result number := 1;
 Begin
   If n < 0 Then
     return null;
 Elsif n = 0, or n = 1 Then
   return 1
 else
   For i IN 2 .. n loop
      Result := result * i ;
      End loop;
     End if;
Return result;
Exception
   when others then
DBMS_output_put_line('An error occur :' ||SQLERRM);
Return null;
End factorial;
/
set serveroutput ON
Declare
    num number := 5;
      Fact number;
END;
```

corresponding book information in library

```
Create or replace Trigger prevent-parent-deletion
Before delete on parent
 For each row
Declare
   child_count number;
Begin
   Select count(*) into child_count from child where parent_id
If child_count >0 then Raise_application_error)
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);
```

Ex.No.: 13		
Date:	1/10	WORKING WITH TRIGGER

TRIGGER

DEFINITION

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are,

- **Trigger statement**: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.
- **Trigger body or trigger action**: It is a PL/SQL block that is executed when the triggering statement is used.
- **Trigger restriction**: Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

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

TYPES OF TRIGGERS

The various types of triggers are as follows,

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

VARIABLES USED IN TRIGGERS

- :new
- :old

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 parent
For each row
Declare
    child_count number;
Begin
  Select count(*) Into child_count from child where parent_id
If child_count > 0 Then Raise_Application_error
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 sample table(
  id number(5) primary key,
name varchar(50) null;
  email varchar2(100) unique
);
create or replace Trigger check_duplicate_email
Before Insert or update on sample table
For each row
   Declare
      duplicate_count Number*
Begin
   Select count(*) INTO duplicate_count
      End IF;

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 or replace trigger restrict_total_sales
Before Insert on sales
For each row
Begin
   If (Select sum (amount) From sales)+ :new.amount >10000.
Raise_application_error (-20002, 'Total exceeds threshold');
End if;
End;
/
```

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 restrict_total_sales
Before Insert on sales
For each row
Begin
   If (Select sum (amount) from sales) + :new.amount >10000
Raise_application_Error (-20002, 'Total exceeds threshold');
End if;
End;
/
```

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 or replace trigger record_user_activity
After insert or update or delete on employees for each row
Begin
  insert into audit_log values (audit_seq.nextval,
case when inserting then 'insert' when updating then 'update'
'Employees', NVL(:OLD.emp_id, :new.emp_id), sysdate, user);
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
amount number(10,2);
    Running_total number(10,2)
  );
  create or replace trigger update_running_total
For each row
Begin
  Select NVL(MAX(running_total),0)+:new.amount INTO:
new.running
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 or replace Trigger Validate_stock_before_order
Before insert on orders
  For each row
Begin
If: New order_quantity > (Select stock_quantity From items
where item_id = : New.item_id
    End if;
    End;
/
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	4
Program/Execution (5)	4
Viva(5)	4
Total (15)	12
Faculty Signature	10

Ex.No.: 14		
Date:	8/10	MONGO DB

MongoDB is a free and open-source cross-platform document-oriented database. Classified as a NoSQL database, MongoDB avoids the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas, making the integration of data in certain types of applications easier and faster.

Create Database using mongosh

After connecting to your database using mongosh, you can see which database you are using by typing db in your terminal.

If you have used the connection string provided from the MongoDB Atlas dashboard, you should be connected to the myFirstDatabase database.

Show all databases

To see all available databases, in your terminal type show dbs.

Notice that myFirstDatabase is not listed. This is because the database is empty. An empty database is essentially non-existant.

Change or Create a Database

You can change or create a new database by typing use then the name of the database.

Create Collection using mongosh

You can create a collection using the createCollection() database method.

Insert Documents

insertOne()

```
db.posts.insertOne({
 title: "Post Title 1",
 body: "Body of post.",
 category: "News",
 likes: 1,
 tags: ["news", "events"],
```

date: Date()

})

Structure of 'restaurants' collection:

EXERCISE 18


```
{
 "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.restaurants.find(

{ $or: [{ cuisine: { $nin: ['American', 'Chinese'] } },

{ name: /^Wil/i }]

})

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.restaurants.find(

{ grades: { $elemMatch: { grade: "A", score: 11, date: new Date("2014-08-11") } } },

{ restaurant_id: 1, name: 1, grades: 1, _id: 0 }

)

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.restaurants.find({"grades.1": {$elemMatch: {grade:"A", score:9, date: new Date("2014-08-11T00:00:00Z")}}}, {restaurant_id:1, name:1, grades:1, _id:0})

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.restaurants.find({"address.coord.1": {$gt: 42, $lt: 52}}, {restaurant_id:1, name:1, address:1, "address.coord":1, _id:0})

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, borough:-1})

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

db.restaurants.find({"address.street": {$exists: false}}).count()==0

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.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.score": {$lt: 5},
    borough: {$in: ["manhattan", "Brooklyn"]},
    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.score": {$lt: 5},
    borough: {$in: ["manhattan", "Brooklyn"]},
    cuisine: {$nin: ["American", "chinese"]}}}
)
```

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}}
    ]}}
)
```

19. 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 and are located in the borough of Manhattan.

```
db.restaurants.find({
borough: "manhattan",
grades: $all [{$elemMatch: {score:2}},{$elemMatch:
{score:6}}])
```

20. 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 and are located in the borough of Manhattan or Brooklyn.

```
db.restaurants.find({
borough: ["manhattan", "brooklyn"]},
grades: {$all: [{$elemMatch: {score:2}},{$elemMatch:
{score:6}}]})
```

21. 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 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.

```
db.restaurants.find({
borough: {$in: ["manhattan", "brooklyn"]},
grades: {$all: [{$elemMatch: {score:2}},{$elemMatch:
{score:6}}]},
cuisine: {$ne: "American"}
})
```

22. 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 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.

```
db.restaurants.find({
grades: { $all: [{ $elemMatch: { score: 2 }}, { $elemMatch: { score: 6 }}]},
borough: { $in: ["Manhattan", "Brooklyn"]},
cuisine: { $in: ["American", "Chinese"]}
}).pretty()
```

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

```
db.restaurants.find({
grades: {
    $elemMatch: {
    $or: [
    { score: 2 }
    { score: 6 }
]}}}).pretty()
```

Sample document of 'movies' collection

```
{
  _id: ObjectId("573a1390f29313caabcd42e8"),
plot: 'A group of bandits stage a brazen train hold-up, only to find a determined posse hot on
their heels.',
genres: [ 'Short', 'Western' ],
runtime: 11,
cast: [
    'A.C. Abadie',
    "Gilbert M. 'Broncho Billy' Anderson",
    'George Barnes',
    'Justus D. Barnes'
  ],
```

released : 1,
directors : 1,
writers : 1,
awards : 1,
year : 1
genres : 1
}}) - d : 0 }}

13. Find all movies with title, languages, released, directors, writers, awards, year, genres, runtime, cast, countries from the 'movies' collection in MongoDB with cast including "Charles Kayser".

db.movies.find({
cast : "Charles Kayser" }}, { title : 1, language : 1, released : 1, directors : 1,
writers : 1, awards : 1, years : 1, genres : 1, runtime : 1, cast : 1, countries : 1, -id :
0 }})

14. Retrieve all movies with title, languages, released, directors, writers, countries from the 'movies' collection in MongoDB that released on May 9, 1893.

db.movies.find({ released : ISO Date("1893-05-09 T00:00:00Z") }),
{ title : 1, languages : 1, released : 1, directors : 1, writers : 1, countries : 1,
id : 0 }})

14. Retrieve all movies with title, languages, released, directors, writers, countries from the 'movies' collection in MongoDB that have a word "scene" in the title.

db.movies.find ({ title : Scene/i }),
{ title : 1, languages : 1, released : 1,
directors : 1, writers : 1, countries : 1, -id : 0 })
/

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	4
Program/Execution (5)	4
Viva(5)	4
Total (15)	12
Faculty Signature	

Ex.No.: 15		
Date:	18/10	OTHER DATABASE OBJECTS

OTHER DATABASE OBJECTS

Objectives

After the completion of this exercise, the students will be able to do the following:
- Create, maintain, and use sequences
- Create and maintain indexes

Database Objects

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.
If you want to improve the performance of some queries, you should consider creating an index. You
can also use indexes to enforce uniqueness on a column or a collection of columns.
You can provide alternative names for objects by using synonyms.

What Is a Sequence?
A sequence:
- Automatically generates unique numbers
- Is a sharable object
- Is typically used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

The CREATE SEQUENCE Statement Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

In the syntax:
sequence is the name of the sequence generator

1. Create sequence dept_ID_seq
Increment By 10
Start with 200
max value 1000
no cache
no cycle;

2. Select
sequence_name,
max_value,
increment_by,
last_number
From
user_sequence
where
sequence_name = 'Dept_ID_Seq';

3. Insert into dept (Dept_ID, Dept_name)
Values (Dept_ID_Seq.Nextval, 'Education');
Insert into dept (Dept_ID, Dept_name)
Values (Dept_ID_SEQ.nextval, 'Health care');

4. Create Index emp_dept_id_idx
on emp (Dept_ID);

5. Select
index_name,
uniqueness
From
user_indexes
where table_name = 'EMP';

Ex.No.: 16

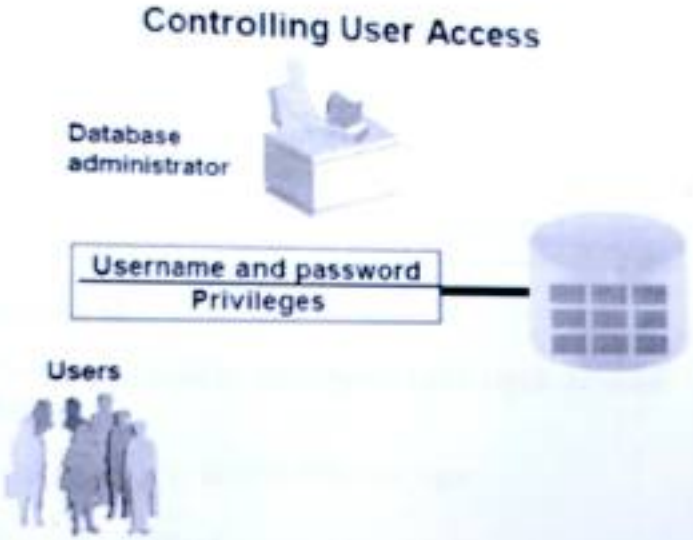
Date: 25/10

CONTROLLING USER ACCESS

Objectives

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

- Create users
- Create roles to ease setup and maintenance of the security model
- Use the GRANT and REVOKE statements to grant and revoke object privileges
- Create and access database links



Controlling User Access

In a multiple-user environment, you want to maintain security of the database access and use. With Oracle server database security, you can do the following:

- Control database access
- Give access to specific objects in the database
- Confirm given and received *privileges* with the Oracle data dictionary
- Create synonyms for database objects

Privileges

- Database security:
  - System security
  - Data security

1. System privilege : The Create session privilege is classified as a system because it allows the user to establish a collection to the database.

2. Grant Create table to Scott;

* Grant : This command is used to provide a privilege to a user
* Create Table : This is the system privilege
* To Scott : This specifies the user to whom the privilege is being granted. You can replace Scott with any valid username.

3. Privileges granted by the owner. Granting privileges are the owner can use the with Grant option clause to allow the grantee to the further pass

4. Create a role :

Create role common_privileges;

Grand privileges :

Grant create session, Create table, Create view to common_privileges

6. Step-1 : Grant access to your department table

Step 2 : Grant query access to his or her departments table

Example : Commands in square

Grant select on department to John;

8. Step-1 : Add new rows :

Insert into departments(department_ID, Department_name)

Values(500, 'education');

9. Select * from user_tables;

Find the Solution for the following:

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?

2. What privilege should a user be given to create tables?

3. If you create a table, who can pass along privileges to other users on your table?

4. You are the DBA. You are creating many users who require the same system privileges. What should you use to make your job easier?

5. What command do you use to change your password? Alter user

6. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

7. Query all the rows in your DEPARTMENTS table. Select * From departments;

8. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team's table.

9. Query the USER_TABLES data dictionary to see information about the tables that you own.

10. Revoke the SELECT privilege on your table from the other team.

11. Remove the row you inserted into the DEPARTMENTS table in step 8 and save the changes.

10. Revoke the select privilege;
    Revoke select on departments from team2;

11. • Delete the rows
    • Commit the changes