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 (
    ID NUMBER (4) NOT NULL,
    Last_name VARCHAR (25),
    First_name VARCHAR (25),
    Userid VARCHAR (25),
    Salary NUMBER (9,2), );
```

O/P: Table created successfully

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);
```

O/P: 2 rows inserted.

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', SUBSTR('Ben', 1, 1) ||
                                                SUBSTR('Biri', 1, 7), 1100);
INSERT INTO MY_EMPLOYEE VALUES(4, 'Newman', 'Chad', SUBSTR('chad', 1, 1)
                                                || SUBSTR('Newman', 1, 7), 750)
```

5. Delete Betty dancs from MY _EMPLOYEE table.

```
DELETE FROM MY_EMPLOYEE
WHERE Last_name = 'Dancs' AND First_name = 'Betty';
```

O/P: 1 row deleted

6. Empty the fourth row of the emp table.

```
UPDATE MY-EMPLOYEE
SET Last-name = NULL , First_name = NULL, User id = NULL , Salary = NULL
WHERE ID = 4 ;
```

7. Make the data additions permanent.

```
COMMIT ;
```

O/P : Transaction committed.

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

```
UPDATE MY-EMPLOYEE
SET Last_name = 'Drexler'
WHERE ID = 3;
```

O/P: 1 row updated

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

```
UPDATE MY_EMPLOYEE SET Salary = 1000 WHERE Salary <900;
```

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

Result :

Thus, creation of base table and DML operations (INSERT, SELECT, UPDATE, DELETE, TRUNCATE, ALTER, DROP) were successfully executed and verified.

Ex.No.: 2		DATA MANIPULATIONS
Date:		

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 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 First_name, Last_name, Salary
FROM EMPLOYEES
WHERE Salary >= 4800;
```

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

```
SELECT First_name, Last_name
FROM EMPLOYEES
WHERE Last_name = 'AUSTIN';
```

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

```
SELECT First_name, Last_name, Department_id
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)

DA = 30% of Basic

HRA = 40% of Basic

PF = 10% of Basic

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

```
INSERT INTO EMP VALUES (1, 'Ravi', 'Manager', 20000, 20000*0.3,
        20000*0.4, 20000*0.1, (20000+20000*0.3+20000*0.4),
        (20000*0.3+20000*0.4)-(20000*0.1));
```

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

```
SELECT EmpName, Basic
FROM EMP
WHERE Basic = (SELECT MIN(Basic) FROM EMP);
```

(c) If Net Pay is less than

```
SELECT EmpName, NetPay
FROM EMP
WHERE NetPay < 10000;
```

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) PRIMARY KEY,
    NAME VARCHAR (25) NOT NULL
);
```

```
CREATE TABLE Emp(
    ID NUMBER (7) PRIMARY KEY,
    Last_Name VARCHAR (25),
    First_Name VARCHAR (25),
    DEPT_ID NUMBER (7),
    FOREIGN KEY (DEPT_ID) REFERENCES DEPT (ID)
);
```

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

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

```
ALTER TABLE EMP MODIFY (LAST_NAME, VARCHAR (50));
```

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 EMPLOYEES 2 AS
SELECT Employee_id, AS ID, First_name, Last_name, Salary,
    Department_id AS Dept_id FROM EMPLOYEES;
```

5 Drop the EMP table.

```
DROP TABLE EMP;
```

6 Rename the EMPLOYEES2 table as EMP.

```
RENAME EMPLOYEES 2 TO EMP;
```

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

```
COMMENT ON TABLE DEPT IS 'Department table';
COMMENT ON TABLE EMP IS 'Employee table';
```

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

```
ALTER TABLE EMP DROP COLUMN First_name;
```

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

Result :

Thus, the EMPLOYEES, EMP, DEPT, LOCATION and other tables were created and various DML/DDL operations such as SELECT, INSERT, UPDATE, DELETE, ALTER, RENAME, DROP were successfully executed.

Ex.No.: 3		WRITING BASIC SQL SELECT STATEMENTS
Date:		

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.

- Keywords cannot be abbreviated or split across lines
- Clauses are usually placed on separate lines
- Indents are sued to enhance readability

Using Arithmetic Expressions

Basic Arithmetic operators like *, /, +, -can be used

Example:1

SELECT last_name, salary, salary+300 FROM employees;

Example:2

SELECT last_name, salary, 12*salary+100 FROM employees;

The statement is not same as
SELECT last_name, salary, 12*(salary+100) FROM employees;

Example:3

SELECT last_name, job_id, salary, commission_pct FROM employees;

Example:4

SELECT last_name, job_id, salary, 12*salary*commission_pct FROM employees;

Using Column Alias

- To rename a column heading with or without AS keyword.

Example:1
SELECT last_name AS Name
FROM employees;
Example: 2
SELECT last_name "Name" salary*12 "Annual Salary "
FROM employees;

Concatenation Operator

- Concatenates columns or character strings to other columns
- Represented by two vertical bars (||)
- Creates a resultant column that is a character expression

Example:

SELECT last_name||job_id AS "EMPLOYEES JOB" FROM employees;

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;
FALSE

Correct :
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, last_name, job_id , hire_date
FROM employees;
```

4. Provide an alias STARTDATE for the hire date.

```
SELECT employee_id , last_name, job_id , 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 || ',' || job_id ||
    ',' || hire_date || ',' || salary || ',' || department_id AS THE_OUTPUT
FROM employees;
```

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

RESULT:

Basic SQL SELECT statements were executed successfully using aliases, arithmetic, DISTINCT, and concatenation.

Ex.No.: 4		WORKING WITH CONSTRAINTS
Date:		

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));

Domain Integrity

This constraint sets a range and any violations that takes place will prevent the user from performing the manipulation that caused the breach.It includes:

NOT NULL Constraint

While creating tables, by default the rows can have null value.the enforcement of not null constraint in a table ensure that the table contains values.

Principle of null values:

- Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will always evaluate to null in any expression.
- When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.
- Not null Integrity constraint cannot be defined using the alter table command when the table contain rows.

Example

CREATE TABLE employees (employee_id number (6), last_name varchar2(25) NOT NULL, salary number(8,2), commission_pct number(2,2), hire_date date constraint emp_hire_date_nn NOT NULL'....);

CHECK

Check constraint can be defined to allow only a particular range of values.when the manipulation violates this constraint,the record will be rejected.Check condition cannot contain sub queries.

CREATE TABLE employees (employee_id number (6), last_name varchar2 (25) NOT NULL, salary number(8,2), commission_pct number(2,2), hire_date date constraint emp_hire_date_nn NOT NULL'...,CONSTRAINT emp_salary_mi CHECK(salary > 0));

Entity Integrity

Maintains uniqueness in a record. An entity represents a table and each row of a table represents an instance of that entity. To identify each row in a table uniquely we need to use this constraint. There are 2 entity constraints:

a) Unique key constraint

It is used to ensure that information in the column for each record is unique, as with telephone or driver's license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value.

If unique key constraint is defined in more than one column i.e., combination of column cannot be specified. Maximum combination of columns that a composite unique key can contain is 16.

Example:

CREATE TABLE employees (employee_id number(6), last_name varchar2(25) NOT NULL,email varchar2(25), salary number(8,2), commission_pct number(2,2), hire_date date constraint emp_hire_date_nn NOT NULL' COSTRAINT emp_email_uk UNIQUE(email));

PRIMARY KEY CONSTRAINT

A primary key avoids duplication of rows and does not allow null values. Can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never be changed and should never be null.
A table should have only one primary key. If a primary key constraint is assigned to more than one column or combination of column is said to be composite primary key, which can contain 16 columns.

Example:

CREATE TABLE employees (employee_id number(6) , last_name varchar2(25) NOT NULL,email varchar2(25), salary number(8,2), commission_pct number(2,2), hire_date date constraint emp_hire_date_nn NOT NULL, Constraint emp_id pk PRIMARY KEY (employee_id),CONSTRAINT emp_email_uk UNIQUE(email));

c) Referential Integrity

It enforces relationship between tables. To establish parent-child relationship between 2 tables having a common column definition, we make use of this constraint. To implement this, we should define the column in the parent table as primary key and same column in the child table as foreign key referring to the corresponding parent entry.

Foreign key

A column or combination of column included in the definition of referential integrity, which would refer to a referenced key.

Referenced key

It is a unique or primary key upon which is defined on a column belonging to the parent table.
Keywords:

FOREIGN KEY: Defines the column in the child table at the table level constraint.

REFERENCES: Identifies the table and column in the parent table.

ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted.

ON DELETE SET NULL: converts dependent foreign key values to null when the parent value is removed.

CREATE TABLE employees (employee_id number(6), last_name varchar2(25) NOT NULL,email varchar2(25), salary number(8,2), commission_pct number(2,2), hire_date date constraint emp_hire_date_nn NOT NULL, Constraint emp_id pk PRIMARY KEY (employee_id), CONSTRAINT emp_email_uk UNIQUE(email), CONSTRAINT emp_dept_fk FOREIGN KEY (department_id) references deparments(dept_id));

ADDING A CONSTRAINT

Use the ALTER to

- Add or Drop a constraint, but not modify the structure
- Enable or Disable the constraints
- Add a not null constraint by using the Modify clause

Syntax

ALTER TABLE table name ADD CONSTRAINT Cons_name type(column name);

Example:

ALTER TABLE employees ADD CONSTRAINT emp_manager_fk FOREIGN KEY (manager_id) REFERENCES employees (employee_id);

DROPPING A CONSTRAINT

Example:

ALTER TABLE employees DROP CONSTRAINT emp_manager_fk;

CASCADE IN DROP

- The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.

Syntax

ALTER TABLE departments DROP PRIMARY KEY|UNIQUE (column)| CONSTRAINT constraint _name CASCADE;

DISABLING CONSTRAINTS

- Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint
- Apply the CASCADE option to disable dependent integrity constraints.

Example

ALTER TABLE employees DISABLE CONSTRAINT emp_emp_id_pk CASCADE;

ENABLING CONSTRAINTS

- Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.

Example

ALTER TABLE employees ENABLE CONSTRAINT emp_emp_id_pk CASCADE;

CASCADING CONSTRAINTS

The CASCADE CONSTRAINTS clause is used along with the DROP column clause.
It drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped Columns.

This clause also drops all multicolumn constraints defined on the dropped column.

Example:

Assume table TEST1 with the following structure

CREATE TABLE test1 (pk number PRIMARY KEY, fk number, col1 number,col2 number, CONTRAINT fk_constraint FOREIGN KEY(fk) references test1, CONSTRAINT ck1 CHECK (pk>0 and col1>0), CONSTRAINT ck2 CHECK (col2>0));

An error is returned for the following statements

ALTER TABLE test1 DROP (pk);

ALTER TABLE test1 DROP (col1);

The above statement can be written with CASCADE CONSTRAINT

ALTER TABLE test 1 DROP(pk) CASCADE CONSTRAINTS;

(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.

```
CREATE TABLE emp (
    id NUMBER (6),
    name VARCHAR2 (20),
    CONSTRAINT my-emp-id-pk 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.

```
CREATE TABLE dept ( id NUMBER (6),
                    dept name VARCHAR2 (20),
                    CONSTRAINT my-dept-id-pk 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(6);

ALTER TABLE emp
ADD CONSTRAINT my-emp-dept-id-fk
FOREIGN KEY (dept-id) REFERENCES 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 comission NUMBER (2,2);

ALTER TABLE emp
ADD CONSTRAINT emp-commission_ck CHECK (comission>0);
```

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

Result:

Integrity constraints (Primary key, foreign key and check) were successfully created and maintained on the EMP and DEPT tables.

Ex.No.: 5		CREATING VIEWS
Date:		

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

CREATE OR REPLACE FORCE/NOFORCE VIEW view_name AS Subquery WITH CHECK OPTION CONSTRAINT constraint WITH READ ONLY CONSTRAINT constraint;

FORCE - Creates the view regardless of whether or not the base tables exist.

NOFORCE - Creates the view only if the ase table exist.

WITH CHECK OPTION CONSTRAINT-specifies that only rows accessible to the view can be inserted or updated.

WITH READ ONLY CONSTRAINT-ensures that no DML operations can be performed on the view.

Example: 1 (Without using Column aliases)
Create a view EMPVU80 that contains details of employees in department80.

Example 2:

CREATE VIEW empvu80 AS SELECT employee_id, last_name, salary FROM employees WHERE department_id=80;

Example:1 (Using column aliases)

```
CREATE VIEW salvu50
  AS  SELECT employee_id,id_number,  last_name NAME, salary *12 ANN_SALARY
FROM employees
    WHERE department_id=50;
```

Retrieving data from a view

Example:
SELECT * from salvu50;

Modifying a view

A view can be altered without dropping, re-creating.

Example: (Simple view)

Modify the EMPVU80 view by using CREATE OR REPLACE.

```
CREATE OR REPLACE VIEW empvu80 (id_number, name, sal, department_id)
 AS     SELECT employee_id,first_name, last_name, salary, department_id
FROM employees
WHERE department_id=80;
```

Example: **(complex view)**

```
CREATE VIEW dept_sum_vu (name, minsal, maxsal,avgsal)
AS      SELECT d.department_name, MIN(e.salary), MAX(e.salary), AVG(e.salary)
FROM employees e, department d
WHERE e.deparment_id=d.deparment_id
GROUP BY d.department_name;
```

Rules for performing DML operations on view

- Can perform operations on simple views
- Cannot remove a row if the view contains the following:
  - Group functions
  - Group By clause
  - Distinct keyword

- Cannot modify data in a view if it contains
  - Group functions
  - Group By clause
  - Distinct keyword
  - Columns contain by expressions

- Cannot add data thr' a view if it contains
  - Group functions
  - Group By clause
  - Distinct keyword
  - Columns contain by expressions
  - NOT NULL columns in the base table that are not selected by the view

Example: **(Using the WITH CHECK OPTION clause)**

```
CREATE OR REPLACE VIEW empvu20
 AS     SELECT *
 FROM employees
 WHERE department_id=20
 WITH CHECK OPTION CONSTRAINT empvu20_ck;
```

Note: Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

Example – (Execute this and note the error)

```
UPDATE empvu20 SET department_id=10 WHERE employee_id=201;
```

Denying DML operations

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 employees_vu AS
SELECT employee_id, last_name AS EMPLOYEE, department_id
FROM employees;
```

2. Display the contents of the EMPLOYEES_VU view.

```
SELECT * FROM employees_vu;
```

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

```
SELECT view_name, text
FROM user_views
WHERE view_name = 'EMPLOYEES_VU';
```

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

```
SELECT EMPLOYEE, department_id
FROM employees_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 OR REPLACE VIEW dept50 (empno, employee, deptno) AS
SELECT employee_id, last_name, department_id
FROM employees WHERE department_id = 50
WITH CHECK OPTION CONSTRAINT dept50_ck;
```

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

```
DESC dept50;
SELECT * FROM dept50;
```

7. Attempt to reassign Matos to department 80.

```
UPDATE dept50
SET deptno = 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 (employee, department, salary, grade) AS
SELECT e.last_name, d.department_name, e.salary, j.grade_level
FROM employees e, departments d, job_grades j
WHERE e.department_id = d.department_id
AND e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

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

Result :

Views were successfully created and tested with constraints. DML operations through views are demonstrated with restrictions (WITH CHECK OPTION and WITH READ ONLY).

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

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

```
FROM employees
WHERE salary<=3000;
```

Other comparison conditions

Operator	Meaning
BETWEEN ...AND...	Between two values
IN	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null values

Example:1

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```

Example:2

```
SELECT employee_id, last_name, salary , manager_id
FROM employees
WHERE manager_id IN (101, 100,201);
```

Example:3

- Use the LIKE condition to perform wildcard searches of valid string values.
- Two symbols can be used to construct the search string
- % denotes zero or more characters
- _ denotes one character

```
SELECT first_name, salary
FROM employees
WHERE first_name LIKE '%s';
```

Example:4

```
SELECT last_name, salary
FROM employees
WHERE last_name LIKE '_o%';
```

Example:5

ESCAPE option-To have an exact match for the actual % and_ characters
To search for the string that contain 'SA_'

```
SELECT employee_id, first_name, salary,job_id
FROM employees
WHERE job_id LIKE '%sa\_%'ESCAPE'\';
```

Test for NULL

- Using IS NULL operator

Example:

```
SELECT employee_id, last_name, salary , manager_id
FROM employees
WHERE manager_id  IS NULL;
```

Logical Conditions

All logical operators can be used.(AND,OR,NOT)

Example:1

```
SELECT employee_id, last_name, salary , job_id
FROM employees
WHERE salary>=10000
AND job_id LIKE '%MAN%';
```

Example:2

```
SELECT employee_id, last_name, salary , job_id
FROM employees
WHERE salary>=10000
OR job_id LIKE '%MAN%';
```

Example:3

```
SELECT employee_id, last_name, salary , job_id
FROM employees
WHERE  job_id NOT IN ('it_prog', st_clerk', sa_rep');
```

Rules of Precedence

Order Evaluated	Operator
1	Arithmetic

2	Concatenation
3	Comparison
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Logical NOT
7	Logical AND
8	Logical OR

Example:1

```
SELECT employee_id, last_name, salary , job_id
FROM employees
WHERE  job_id ='sa_rep'
OR job_id='ad_pres'
AND salary>15000;
```

Example:2

```
SELECT employee_id, last_name, salary , job_id
FROM employees
WHERE  (job_id ='sa_rep'
OR job_id='ad_pres')
AND salary>15000;
```

Sorting the rows

Using ORDER BY Clause

ASC-Ascending Order,Default

DESC-Descending order

Example:1

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

Example:2

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

Example:3

Sorting by column alias

```
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-MAY-1998'
ORDER BY hire_date ASC;
```

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

```
SELECT 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 AS EMPLOYEE, salary AS "MONTHLY SALARY"
FROM employees WHERE salary BETWEEN 5000 AND 12000
AND department_id IN (20, 50)
ORDER BY last_name;
```

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)

```
SELECT last_name, salary, commission_pct
FROM employees
WHERE commission_pct IS NOT NULL
ORDER BY salary DESC, commission_pct DESC;
```

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

```
SELECT 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)

```
SELECT 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)

```
SELECT 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)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Result:

Queries using WHERE and ORDER BY clauses were executed successfully to restrict and sort data.

Ex.No.: 7		USING SET OPERATORS
Date:		

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.

• By default, the output is sorted in ascending order of the first column of the SELECT clause.

Example:

Display the current and previous job details of all employees. Display each employee only once.

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

Example:

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

UNION ALL Operator

Guidelines

The guidelines for UNION and UNION ALL are the same, with the following two exceptions that pertain to UNION ALL:

• Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default.
• The DISTINCT keyword cannot be used.

Example:

Display the current and previous departments of all employees.

```
SELECT employee_id, job_id, department_id
FROM employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

INTERSECT 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.
• Reversing the order of the intersected tables does not alter the result.
• INTERSECT does not ignore NULL values.

Example:

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 department_id FROM departments
MINUS
SELECT department_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 JOIN departments d
ON c.country-id = d.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, department_id FROM employees
WHERE department_id = 10 UNION
SELECT job_id, department_id FROM employees
WHERE department_id = 50 UNION
SELECT job_id, department_id FROM employees
WHERE department_id
= 20;
```

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 employee_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, department_id FROM employees UNION
SELECT department_name, department_id FROM departments;
```

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

Result:

Set operators (UNION, UNION ALL, INTERSECT, MINUS) were successfully used to combine and compare query results.

Ex.No.: 8		WORKING WITH MULTIPLE TABLES
Date:		

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
```

```
FROM table1, table2
WHERE table1.column1 = table2.column2;
```

Write the join condition in the WHERE clause.
• Prefix the column name with the table name when the same column name appears in more than one
table.

Guidelines

• When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
• If the same column name appears in more than one table, the column name must be prefixed with the table name.
• To join n tables together, you need a minimum of n-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row

What is an Equijoin?

To determine an employee's department name, you compare the value in the DEPARTMENT_ID
column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table.
The relationship between the EMPLOYEES and DEPARTMENTS tables is an equijoin—that is, values
in the DEPARTMENT_ID column on both tables must be equal. Frequently, this type of join involves
primary and foreign key complements.
Note: Equijoins are also called simple joins or inner joins

```
SELECTemployees.employee_id,employees.last_name,employees.department_id,
departments.department_id,departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

Additional Search Conditions

Using the AND Operator

Example:

To display employee Matos'department number and department name, you need an additional condition in the WHERE clause.

```
SELECT last_name, employees.department_id,
department_name
FROM   employees, departments
WHERE  employees.department_id = departments.department_id AND   last_name = 'Matos';
```

Qualifying Ambiguous

Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Improve performance by using table prefixes.
- Distinguish columns that have identical names but reside in different tables by using column aliases.

Using Table Aliases

- Simplify queries by using table aliases.
- Improve performance by using table prefixes

Example:

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM   employees e , departments d
WHERE  e.department_id = d.department_id;
```

Joining More than Two Tables

To join n tables together, you need a minimum of n-1 join conditions. For example, to join three tables, a minimum of two joins is required.

Example:

To display the last name, the department name, and the city for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables.

```
SELECT e.last_name, d.department_name, l.city
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

Non-Equijoins

A non-equijoin is a join condition containing something other than an equality operator.The relationship between the EMPLOYEES table and the JOB_GRADES table has an example of a non-equijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST_SALARY and HIGHEST_SALARY columns of the JOB_GRADES table. The relationship is obtained using an operator other than equals (=).

Example:

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary
BETWEEN j.lowest_sal AND j.highest_sal;
```

Outer Joins

Syntax

• You use an outer join to also see rows that do not meet the join condition.
• The Outer join operator is the plus sign (+).

```
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column(+) = table2.column;
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column = table2.column(+);
```

The missing rows can be returned if an outer join operator is used in the join condition. The operator
is a plus sign enclosed in parentheses (+), and it is placed on the "side" of the join that is deficient in
information. This operator has the effect of creating one or more null rows, to which one or more rows
from the nondeficient table can be joined.

Example:

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

Outer Join Restrictions

• The outer join operator can appear on only one side of the expression—the side that has information missing. It returns those rows from one table that have no direct match in the other table.
• A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator

Self Join

Sometimes you need to join a table to itself.

Example:

To find the name of each employee's manager, you need to join the EMPLOYEES table to itself, or perform a self join.

```
SELECT worker.last_name || ' works for '
|| manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id ;
```

Use a join to query data from more than one table.

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

```
FROM table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
ON(table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name = table2.column_name)];
```

In the syntax:
table1.column Denotes the table and column from which data is retrieved
CROSS JOIN Returns a Cartesian product from the two tables
NATURAL JOIN Joins two tables based on the same column name
JOIN table USING column_name Performs an equijoin based on the column name
JOIN table ON table1.column_name Performs an equijoin based on the condition in the ON clause
= table2.column_name

LEFT/RIGHT/FULL OUTER

Creating Cross Joins

- The CROSS JOIN clause produces the crossproduct of two tables.
- This is the same as a Cartesian product between the two tables.

Example:

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
SELECT last_name, department_name
FROM   employees, departments;
```

Creating Natural Joins

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

Example:

```
SELECT department_id, department_name,
location_id, city
FROM   departments
NATURAL JOIN locations ;
```

LOCATIONS table is joined to the DEPARTMENT table by the LOCATION_ID column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

Example:

```
SELECT  department_id, department_name,
location_id, city
FROM    departments
NATURAL JOIN locations
WHERE   department_id IN (20, 50);
```

Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, the NATURAL JOIN clause can be modified with the USING clause to specify the columns that should be used for an equijoin.
- Use the USING clause to match only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.
- The NATURAL JOIN and USING clauses are mutually exclusive.

Example:

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d USING (location_id)
WHERE  location_id = 1400;
```

EXAMPLE:

```
SELECT e.employee_id, e.last_name, d.location_id
FROM   employees e JOIN departments d
USING (department_id) ;
```

Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- To specify arbitrary conditions or specify columns to join, the ON clause is used.
- The join condition is separated from other searchconditions.
- The ON clause makes code easy to understand.

Example:

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

EXAMPLE:

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
```

```
ON    (e.manager_id = m.employee_id);
```

INNER Versus OUTER Joins

• A join between two tables that returns the results of the inner join as well as unmatched rows left (or right) tables is a left (or right) outer join.

• A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.

LEFT OUTER JOIN
Example:

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

Example of LEFT OUTER JOIN
This query retrieves all rows in the EMPLOYEES table, which is the left table even if there is no match in the DEPARTMENTS table.

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;
```

RIGHT OUTER JOIN
Example:

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

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

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
JOIN departments d ON 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 e.job_id, d.location_id
FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE e.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
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
WHERE e.commission_pct IS NOT NULL;
```

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

```
SELECT e.last_name, d.department_name FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE e.last_name LIKE '%a%';
```

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

```
SELECT e.last_name, e.job_id, e.department_id, d.departmentname
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
WHERE l.city = '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 e.last_name AS Employee, e.employee_id AS Emp#,
       m.last_name AS Manager, m.employee_id AS Mgr#
FROM employees e
JOIN employees m ON e.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 e.last_name AS employee, e.employee_id AS Emp#,
       m.last_name AS Manager, m.employee_id AS Mgr#
FROM employees e
LEFT JOIN employees m ON e.manager_id = m.employee_id
ORDER BY e.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.last_name AS Employee, e.department_id AS Dept#,
       d.department_name AS Department
FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE e.department_id = ( SELECT department_id FROM employees );
```

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
```

```
JOIN departments d ON e.department_id = d.department_id
JOIN job_grades j ON 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 last_name, hire_date
FROM employees
WHERE hire_date > ( SELECT hire_date FROM employees WHERE
                    last_name = 'Davies');
```

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 e.last_name AS Employee, e.hire_date AS "Emp Hired",
       m.last_name AS Manager, m.hire_date AS "Mgr Hired"
FROM employees e
JOIN employees m ON e.manager_id = m.manager_id
WHERE e.hire_date < m.hire_date;
```

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

Result:

Queries using equijoins, non-equijoins, outer joins, self joins and cross joins were executed successfully, displaying combined data from multiple tables.

Ex.No.: 9		SUB QUERIES
Date:		

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

(>, =, >=, <, <>, <=) and multiple-row operators (IN, ANY, ALL). statement. The subquery generally executes first, and its output is used to complete the query condition for the main (or outer) query

Using a Subquery

SELECT last_name FROM employees WHERE salary > (SELECT salary FROM employees WHERE last_name = 'Abel');

The inner query determines the salary of employee Abel. The outer query takes the result of the inner query and uses this result to display all the employees who earn more than this amount.

Guidelines for Using Subqueries

• Enclose subqueries in parentheses.

• Place subqueries on the right side of the comparison condition.

• The ORDER BY clause in the subquery is not needed unless you are performing Top-N analysis.

• Use single-row operators with single-row

subqueries, and use multiple-row operators with multiple-row subqueries.

Types of Subqueries

• Single-row subqueries: Queries that return only one row from the inner SELECT statement.

• Multiple-row subqueries: Queries that return more than one row from the inner SELECT statement.

Single-Row Subqueries

• Return only one row

• Use single-row comparison operators

Example

Display the employees whose job ID is the same as that of employee 141:

SELECT last_name, job_id FROM employees WHERE job_id = (SELECT job_id FROM employees
WHERE employee_id = 141);

Displays employees whose job ID is the same as that of employee 141 and whose salary is greater
than that of employee 143.

```
SELECT last_name, job_id, salary FROM employeesWHERE job_id =(SELECT job_id FROM
employees WHERE employee_id = 141) AND salary > (SELECT salary FROM employees
WHERE employee_id = 143);
```

Using Group Functions in a Subquery

Displays the employee last name, job ID, and salary of all employees whose salary is equal to the minimum salary. The MIN group function returns a single value (2500) to the outer query.

```
SELECT last_name, job_id, salary FROM employees WHERE salary = (SELECT MIN(salary)
FROM employees);
```

The HAVING Clause with Subqueries

- The Oracle server executes subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.

Displays all the departments that have a minimum salary greater than that of department 50.

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) >
(SELECT MIN(salary)
FROM employees
WHERE department_id = 50);
```

Example

Find the job with the lowest average salary.

```
SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
FROM employees
GROUP BY job_id);
```

What Is Wrong in this Statements?

```
SELECT employee_id, last_name
FROM employees
```

WHERE salary =(SELECT MIN(salary) FROM employees GROUP BY department_id);
Will This Statement Return Rows?
SELECT last_name, job_id
FROM employees
WHERE job_id =(SELECT job_id FROM employees WHERE last_name = 'Haas');

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Example

Find the employees who earn the same salary as the minimum salary for each department.

SELECT last_name, salary, department_id FROM employees WHERE salary IN (SELECT MIN(salary)
FROM employees GROUP BY department_id);

Using the ANY Operator in Multiple-Row Subqueries

SELECT employee_id, last_name, job_id, salary FROM employees WHERE salary < ANY (SELECT salary FROM employees WHERE job_id = 'IT_PROG') AND job_id <> 'IT_PROG';

Displays employees who are not IT programmers and whose salary is less than that of any IT programmer. The maximum salary that a programmer earns is $9,000.

< ANY means less than the maximum. >ANY means more than the minimum. =ANY is equivalent to IN.

Using the ALL Operator in Multiple-Row Subqueries

SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL (SELECT salary FROM employees WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
Displays employees whose salary is less than the salary of all employees with a job ID of IT_PROG and whose job is not IT_PROG.

- ALL means more than the maximum, and <ALL means less than the minimum.

The NOT operator can be used with IN, ANY, and ALL operators.

Null Values in a Subquery

SELECT emp.last_name FROM employees emp

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 = '&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 ASC;
```

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.

```
SELECT last_name, department_id, job_id
FROM employees
WHERE department_id IN (
    SELECT department_id FROM departments
    WHERE location_id = 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 = (
    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 employees WHERE department_id = (
    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 employees
WHERE salary > (SELECT AVG(salary) FROM employees)
AND department_id IN (
    SELECT department_id
    FROM employees
    WHERE last_name LIKE '%u%');
```

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

Result :

Queries using single-row and multi-row subqueries were executed successfully, retreiving employee information as per HR requirements.

Ex.No.: 10		AGGREGATING DATA USING GROUP FUNCTIONS
Date:		

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 BY *column*]
[ORDER BY *column*];

Guidelines for Using Group Functions

• DISTINCT makes the function consider only nonduplicate values; ALL makes it consider every value, including duplicates. The default is ALL and therefore does not need to be specified.

• The data types for the functions with an expr argument may be CHAR, VARCHAR2, NUMBER, or DATE.

• All group functions ignore null values.

Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),
MIN(salary), SUM(salary)
FROM employees
WHERE job_id LIKE '%REP%';
```

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM employees;
```

You can use the MAX and MIN functions for numeric, character, and date data types. example displays the most junior and most senior employees.

The following example displays the employee last name that is first and the employee last name that is last in an alphabetized list of all employees:

```
SELECT MIN(last_name), MAX(last_name)
FROM employees;
```

Note: The AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types. MAX and MIN cannot be used with LOB or LONG data types.

Using the COUNT Function

COUNT(*) returns the number of rows in a table:

```
SELECT COUNT(*)
FROM employees
```

WHERE department_id = 50;
COUNT(*expr*) returns the number of rows with nonnull values for the *expr*:
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;

Using the DISTINCT Keyword

• COUNT(DISTINCT expr) returns the number of distinct non-null values of the *expr*.

• To display the number of distinct department values in the EMPLOYEES table:

SELECT COUNT(DISTINCT department_id) FROM employees;

Use the DISTINCT keyword to suppress the counting of any duplicate values in a column.

Group Functions and Null Values

Group functions ignore null values in the column:

SELECT AVG(commission_pct)
FROM employees;

The NVL function forces group functions to include null values:

SELECT AVG(NVL(commission_pct, 0))
FROM employees;

Creating Groups of Data

To divide the table of information into smaller groups. This can be done by using the GROUP BY clause.

GROUP BY Clause Syntax

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

In the syntax:
group_by_expression specifies columns whose values determine the basis for grouping rows

Guidelines

• If you include a group function in a SELECT clause, you cannot select individual results as well, *unless* the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column list in the GROUP BY clause.

• Using a WHERE clause, you can exclude rows before dividing them into groups.

• You must include the *columns* in the GROUP BY clause.

• You cannot use a column alias in the GROUP BY clause.

Using the GROUP BY Clause

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id ;
```

The GROUP BY column does not have to be in the SELECT list.

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

You can use the group function in the ORDER BY clause:

```
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id ORDER
BY AVG(salary);
```

Grouping by More Than One Column

```
SELECT department_id dept_id, job_id, SUM(salary) FROM employees
GROUP BY department_id, job_id ;
```

Illegal Queries Using Group Functions

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP

BY clause:

```
SELECT department_id, COUNT(last_name) FROM employees;
```

You can correct the error by adding the GROUP BY clause:

SELECT department_id, count(last_name) FROM employees GROUP BY department_id;

You cannot use the WHERE clause to restrict groups.

- You use the HAVING clause to restrict groups.

- You cannot use group functions in the WHERE clause.

SELECT department_id, AVG(salary) FROM employees WHERE AVG(salary) > 8000
GROUP BY department_id;
You can correct the error in the example by using the HAVING clause to restrict groups:

SELECT department_id, AVG(salary) FROM employees
HAVING AVG(salary) > 8000 GROUP BY department_id;

Restricting Group Results

With the HAVING Clause .When you use the HAVING clause, the Oracle server restricts groups as follows:
1. Rows are grouped.
2. The group function is applied.
3. Groups matching the HAVING clause are displayed.

Using the HAVING Clause

SELECT department_id, MAX(salary) FROM employees
GROUP BY department_idHAVING MAX(salary)>10000 ;

The following example displays the department numbers and average salaries for those departments with a maximum salary that is greater than $10,000:

SELECT department_id, AVG(salary) FROM employees GROUP BY department_id
HAVING max(salary)>10000;

Example displays the job ID and total monthly salary for each job that has a total payroll exceeding $13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

SELECT job_id, SUM(salary) PAYROLL FROM employees WHERE job_id NOT LIKE '%REP%'
GROUP BY job_id HAVING SUM(salary) > 13000 ORDER BY SUM(salary);

Nesting Group Functions

Display the maximum average salary:

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 ROUND(MAX(SALARY)) AS Maximum,
       ROUND(MIN(salary)) AS Minimum,
       ROUND(SUM(salary)) AS Sum,
       ROUND(AVG(salary)) AS Average
FROM employees;
```

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

```
SELECT job-id,
    MIN(salary) AS Minimum,
    MAX(salary) AS Maximum,
    SUM(salary) AS Sum,
    ROUND(AVG(salary)) AS Average
FROM employees
GROUP BY 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 COUNT (*) AS Num_of_People
FROM employees
WHERE job_id = '&job_title';
```

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 (DISTINCT manager_id) AS Number_of_managers
FROM employees;
```

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 manager_id, MIN (salary) AS Lowest_Salary
FROM employees WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING MIN (salary) > 6000
ORDER BY Lowest_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_Employees,
    SUM(CASE WHEN TO_CHAR(hire_date, 'YYYY')='1995' THEN 1 ELSE 0 END)
                                AS Hired_1995,
SUM (CASE WHEN TO_CHAR(hire_date, 'YYYY')='1996' THEN 1 ELSE 0 END)
                                AS Hired_1996,
SUM(CASE WHEN TO_CHAR(hire_date, 'YYYY')= '1997' THEN 1 ELSE 0 END)
                                AS Hired_1997,
```

```
SUM(CASE WHEN TO_CHAR(hire_date, 'YYYY') = '1998' THEN 1 ELSE 0
                END) AS Hired_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 job_id,
  SUM (CASE WHEN department_id = 20 THEN salary ELSE 0 END) AS Dept20,
  SUM(CASE WHEN department_id = 50 THEN salary ELSE 0 END) AS Dept 50,
  SUM( CASE WHEN department_id = 80 THEN salary ELSE 0 END) AS Dept 80,
  SUM( CASE WHEN department_id = 90 THEN salary ELSE 0 END) AS Dept 90,
  SUM(salary) AS Total FROM employees GROUP BY 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 || '-' || l.city AS Name_location,
    COUNT(e.employee_id) AS Number_of_people,
    ROUND(AVG(e.salary), 2) AS Salary
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
GROUP BY d.department_name, l.city;
```

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

Result :

Queries using group functions, GROUP BY, and HAVING were successfully, ~~ex~~generating reports as per HR department requirements.

PROGRAM 1

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

```
SET SERVEROUTPUT ON;
DECLARE
  v_emp_id employees.employee_id %TYPE := 110;
  v_incentive NUMBER (8,2);
  v_salary employees.salary %TYPE ;
BEGIN
  SELECT salary INTO v_salary FROM employees WHERE
                                  employee_id = v_emp_id;
  v_incentive := v_salary * 0.10;
  DBMS_OUTPUT.PUT_LINE ('Employee ID: ' || v_emp_id);
  DBMS_OUTPUT.PUT_LINE ('Incentive: ' || v_incentive);
END;
/
```

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
  "Name" VARCHAR(20) := 'Joyce';
BEGIN
  DBMS_OUTPUT.PUT_LINE (name);
END;
/
```

PROGRAM 3

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

```
SET SERVEROUTPUT ON;
DECLARE
  v_emp_id employees.employee_id %TYPE := 122;
BEGIN
  UPDATE employees
  SET Salary = salary * 1.10
  WHERE employee_id = v_emp_id;
  IF SQL %ROWCOUNT > 0 THEN
    DBMS_OUTPUT.PUTLINE ('Salary updated for Employee '||v_emp_id);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Employee not found');
  END IF;
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;
CREATE OR REPLACE PROCEDURE check_null_values IS
  v_name VARCHAR2 (20);
  v_id NUMBER := 101;
BEGIN
  IF v_name IS NULL AND v_id IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE ('Condition TRUE');
  ELSE
    DBMS_OUTPUT.PUT_LINE ('condition FALSE');
  END IF;
END;
/
BEGIN
    check_null_values;
END;
/
```

PROGRAM 5

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

```
SET SERVEROUTPUT ON;
DECLARE
  v_name employees.first_name %TYPE;
BEGIN
  FOR rec IN (SELECT first_name FROM employees WHERE first_name LIKE 'S%')
  LOOP
    DBMS_OUTPUT.PUT_LINE('Name starting with S: '||rec.first_name);
  END LOOP;
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.

```
SET SERVEROUTPUT ON;
DECLARE
  num1 NUMBER := &num1;
  num2 NUMBER := &num2;
  num_small NUMBER;
  num_large NUMBER;
BEGIN
  IF num1<num2 THEN
     num_small := num1;
     num_large := num2;
  ELSE
     num_small := num2;
     num_large := num1;
  END IF;
  DBMS_OUTPUT.PUT_LINE ('Small number: '||num_small);
  DBMS_OUTPUT.PUT_LINE ('Large number: '||num_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
   v_salary employees.salary %TYPE;
   v_incentive NUMBER;
BEGIN
   SELECT salary INTO v_salary FROM employees WHERE employee_id = emp_id;
   v_incentive := v_salary * 0.10;
   UPDATE employees SET salary = salary + v_incentive WHERE employee_id
                                                          = emp_id;
   IF SQL %ROWCOUNT > 0 THEN
      DBMS_OUTPUT.PUT_LINE ('Record updated. Incentive added');
   ELSE
      DBMS_OUTPUT.PUT_LINE ('No Record updated');
   END IF;
END;
/
```

PROGRAM 8

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

```
SET SERVEROUTPUT ON;
DECLARE
   v_sale NUMBER := &v_sale;
   v_incentive NUMBER;
BEGIN
   IF v_sale >= 100000 THEN
      v_incentive := v_sale * 0.10;
   ELSIF v_sale >= 50000 THEN
      v_incentive := v_sale * 0.05;
   ELSE
      v_incentive := v_sale * 0.02;
   END IF;
   DBMS_OUTPUT.PUTLINE ('Sale: ' || v_sale);
   DBMS_OUTPUT.PUTLINE ('Incentive: ' || v_incentive);
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 SERVEROUTPUT ON;
DECLARE
   v_count NUMBER;
   v_vacancies NUMBER := 45;
BEGIN
  SELECT COUNT (*) INTO v_count FROM employees WHERE department_id = 50;
  IF v_count < v_vacancies THEN
    DBMS_OUTPUT.PUT_LINE ('Vacancies available : ' ||(v_vacancies-v_count))
  ELSE
    DBMS_OUTPUT.PUT_LINE ('No vacancies available');
  END IF;
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 SERVEROUTPUT ON;
DECLARE
   v_dept_id NUMBER := &v_dept_id;
   v_count NUMBER;
   v_vacancies NUMBER := 45;
BEGIN
   SELECT COUNT (*) INTO v_count FROM employees WHERE dept_id =
                                                        v_dept_id
   IF v_count < v_vacancies THEN
     DBMS_OUTPUT.PUT_LINE ('Vacancies available in Department '
                          ||v_dept_id ||':' || (v_vacancies-v_count));
   ELSE
     DBMS_OUTPUT.PUT_LINE( 'No vacancies in Department '
                                                  ||v_dept_id);
   END IF;
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, first_name, job_id,
              hire_date FROM employees)
  LOOP
    DBMS_OUTPUT.PUT_LINE('ID : '||rec.employee_id||
    'Name : '||rec.first_name||'Job : '||rec.job_id||
    'Hiredate : '||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.first_name, d.department_name
      FROM employees e
      JOIN departments d ON e.department_id = d.department_id
  )
  LOOP
    DBMS_OUTPUT.PUT_LINE('ID : '||rec.employee_id||'Name : '||
        rec.first_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).
  LOOP
    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 employee-id, start-date FROM job-history)
  LOOP
    DBMS_OUTPUT.PUT_LINE('Employee ID: '||rec.employee-id||
                         'Job Start Date: '||rec.start-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 employee-id, end-date FROM job-history)
  LOOP
    DBMS_OUTPUT.PUT_LINE('Employee ID: '||rec.employee-id||
                         'Job End Date: '||rec.end-date)
  END LOOP;
END;
/
```

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

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

```
CREATE OR REPLACE FUNCTION get_factorial (n NUMBER)
RETURN NUMBER
IS
  fact NUMBER := 1;
  i NUMBER;
BEGIN
  IF n < 0 THEN
     RETURN NULL;
  END IF;
  i := n;
  WHILE i > 0 LOOP
     fact := fact * i;
     i := i-1;
  END LOOP;
  RETURN fact;
END;
/
SET SERVEROUTPUT ON;
DECLARE
     input_num NUMBER := 7
     result NUMBER;
BEGIN
     result := get_factorial (input_num);
     dbms_output.put_line ('Factorial of '||input_num||' is'||
                                                  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 get_book_info(
    p_book_id IN NUMBER,
    p_title OUT VARCHAR2,
    p_author OUT VARCHAR2,
    p_copies IN OUT NUMBER
  )
IS v_db_copies NUMBER;

BEGIN
    SELECT title, author, copies
    INTO p_title, p_author, v_db_copies
    FROM library_books
    WHERE book_id = p_book_id;
    p_copies := v_db_copies - 1;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_title := NULL;
        p_author := NULL;
        p_copies := NULL;
        dbms_output.put_line('Book not found.');
END;
/
```

```
SET SERVEROUTPUT ON;
DECLARE
  v_book_id NUMBER := 2;
  v_title VARCHAR(100);
  v_author VARCHAR(100);
  v_copies NUMBER := 0;
BEGIN
  get_book_info (v_bookid,
      v_title, v_author, v_copies
  dbms_output.put_line('Book ID:'
      || v_bookid);
  dbms_output.put_line('Title:'||
      v_title);
  dbms_output.put_line('Author
      || v_author);
  dbms_output.put_line('copies
      left:'||v_copies);
END;
```

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

```
declare
cursor cenl is select eid,sal from ssempp where dno=11;
ecode ssempp.eid%type;
esal empp.sal%type;
begin
open cenl;
loop
fetch cenl into ecode,esal;
exit when cenl%notfound;
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 trg_prevent_parent_delete
BEFORE DELETE ON parent_table
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_count
    FROM child_table
    WHERE parent_id = :OLD.parent_id;
    IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(
            -20001,
            'cannot delete parent record : child records exists!
        );
    END IF;
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 OR REPLACE TRIGGER trg_prevent_duplicate_rollno
BEFORE INSERT OR UPDATE ON student
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO v_count
    FROM student
    WHERE roll_no = :NEW.roll_no
        AND (:OLD.roll_no IS NULL OR roll_no != :OLD.roll_no);
    IF v_count > 0 Then
        RAISE_APPLICATION_ERROR(
            -20002,
            'Duplicate value not allowed for ROLL_NO.'
        );
    END IF;
```

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 trg_limit_total_amount
BEFORE INSERT ON orders
FOR EACH ROW
DECLARE
    V_total NUMBER;
    V_threshold CONSTANT NUMBER := 100000;
BEGIN
    SELECT NVL(SUM(amount),0)
    INTO V_total;
    FROM orders;
    V_total := V_total + :NEW.amount;
    IF V_total > V_threshold THEN
        RAISE_APPLICATION_ERROR(
            -20003, 'INSERT restricted: TOTAL AMOUNT exceeds allowed Thr
            );
    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 trg_employee_audit
AFTER UPDATE ON employee
FOR EACH ROW
BEGIN
    IF :OLD.salary != :NEW.salary THEN
        INSERT INTO employee_audit (
            emp_id, column_name, old_value, new_value, changed_by,
                                                       changed_on
        )
        VALUES (
            :OLD.emp_id,
            'SALARY',
            TO_CHAR(:OLD.salary),
            TO_CHAR(:NEW.salary),
            USER, SYSDATE
        );
    ENDIF;
    IF :OLD.department_id != :NEW.department_id THEN
        INSERT INTO employee_audit(
            emp_id, column_name, old_value, new_value, changed_by,
```

```
VALUES (
    : OLD. emp_id,
    'DEPARTMENT_ID';
    TO_CHAR ( : OLD. department_id ),
    TO_CHAR ( : NEW. department_id ),
    USER,
    SYSDATE
);
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 trg_employee_audit_log
AFTER INSERT OR UPDATE OR DELETE ON employee
FOR EACH ROW
DECLARE
    V_old VARCHAR2(4000);
    V_new VARCHAR2(4000);
BEGIN
    IF DELETING OR UPDATING THEN
        V_old := 'ID =' || : OLD. emp_id||
                 ', NAME= ' || : OLD. empname||
                 ', SALARY = ' || : OLD. salary||
                 ', DEPT = ' || : OLD. department_id;
    END IF;
    IF INSERTING OR UPDATING THEN
        V_new := 'ID =' || : NEW. emp_id||
                 ', NAME = ' || : NEW. empname||
                 ', SALARY =' || : NEW. salary||
                 ', DEPT= ' || : NEW. department_id;
    END IF;
    INSERT INTO audit_log (
        table_name,
        operation,
        user_name,
        old_value,
        new_value,
        action_date
    )
    VALUES (
        'EMPLOYEE',
        CASE
        WHEN INSERTING THEN 'INSERT'
        WHEN UPDATING THEN 'UPDATE'
        WHEN DELETING THEN 'DELETE'
        END, USER, V_old, V_new, SYSDATE
    );
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 OR REPLACE TRIGGER trg_sales_running_total
BEFORE INSERT ON Sales
FOR EACH ROW
DECLARE
    V_total NUMBER;
BEGIN
    SELECT NVL(SUM(amount), 0)
    INTO V_total
    FROM Sales;
    : NEW. running_total := V_total + : NEW. amount;
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 TRIGGERtrg_validate_item_stock
BEFORE INSERT ON orders
FOR EACH ROW
DECLARE
    v_stock NUMBER;
     v_reserved NUMBER;
     v_available NUMBER;
BEGIN
    SELECT stock_qty
    INTO v_stock
     FROM items
     WHERE item_id = :NEW.item_id;
     SELECT NVL(SUM(order_qty),0)
     INTO v_reserved
     FROM orders
     WHERE item_id = :NEW.item_id
          AND status = 'PENDING';
     v_available := v_stock - v_reserved;
     IF :NEW.order_qty > v_available
                                THEN
        RAISE_APPLICATION_ERROR(
            -20010,
            'order cannot be placed:
                insufficient stock available
            );
        END IF;
    END;
/
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (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. restaurants. find (
  { $or: [{
      cuisine: {
        $nin: ["American", "chinese"] } },
    { name: { $regex : /^wil/i } } ] }, { restaurant_id:1, name
    borough:
    cuisine: 1
    _id: 0
  } );
```

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 : ISODate ("2014-08-11T00:00:00Z")
      } } } ,
  { restaurant_id:1,
    name: 1,
    grades: 1,
    _id: 0
  } );
```

```
db.restaurants.find(
{
"grades.1.grade" : "A",
"grades.1.score": 9,
"grades.1.date": ISODate("2014-08-11T00:00:00Z")
},
{ 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".

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,
$lte: 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(
{
"address.coord.1": {
$gt: 42,
$lte: 52
},
},
{ restaurant_id: 1,
name: 1,
address: 1,
"address.coord": 1,
_id: 0 });
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 }
});
```

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,
    _id:0
  });
```

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: /mon/i } },
  { name:1,
    borough:1,
    "address.coord":1,
    cuisine:1,
    _id:0
  });
```

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/i } },
  { name:1,
    borough:1,
    "address.coord":1,
    cuisine:1,
    _id:0
  });
```

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.score": { $lt:5 } },
  { restaurant_id:1,
    name:1,
    grades:1,
    _id:0 });
```

14. 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.

```
db.restaurants.find(
  { borough: "Manhattan",
    "grades.score": { $lt:5 } },
  { restaurant_id:1,
    name:1,
    borough:1,
    grades:1,
```

```
_id : 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(
{
borough : { $in : [ "Manhattan", "Brooklyn" ] },
"grades.score" : { $lt : 5 }
},
{ restaurant_id : 1,
name : 1,
borough : 1,
grades : 1,
_id : 0
}
);
```

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(
{
borough : { $in : [ "Manhattan", "Brooklyn" ] },
"grades.score" : { $lt : 5 },
cuisine : { $ne : "American" }
},
{ restaurant_id : 1,
name : 1,
borough : 1,
cuisine : 1,
grades : 1,
_id : 0
},
);
```

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(
{
borough : { $in : [ "Manhattan", "Brooklyn" ] },
"grades.score" : { $lt : 5 },
cuisine : { $nin : [ "American", "chinese" ] }
},
{ restaurant_id : 1,
name : 1,
borough : 1,
cuisine : 1,
grades : 1,
_id : 0
}
);
```

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.score" : 2,
"grades.score" : 6
},
{ restaurant_id : 1,
name : 1,
grades : 1,
```

```
_id:0
},
);
```

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.score": 2,
  "grades.score":6
  },
  {
   restaurant_id:1,
   name:1, borough:1,
   grades:1,
   _id:0 });
```

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: { $in: ["Manhattan","Brooklyn"] },
   "grades.score": 2,
   "grades.score": 6
  },
  { restaurant_id:1,
    name:1,
    borough:1,
    grades:1,
    _id:0
  });
```

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"] },
   cuisine: { $ne: "American" },
   "grades.score": 2,
   "grades.score": 6
   },
   {
    restaurant_id:1,
    name:1,
    borough:1,
    cuisine:1,
```

```
grades:1,
_id:0
}
);
```

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(
{borough:{$in:["Manhattan","Brooklyn"]},
cuisine:{$nin:["American","chinese"]},
"grades.score":2,
"grades.score":6
},
{restaurant_id:1,
name:1,
borough:1,
cuisine:1,
grades:1,
_id:0});
```

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(
{$or:[
{"grades.score":2},
{"grades.score":6}
]},
{restaurant_id:1,
name:1,
grades:1,
_id:0
}
);
```

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'
],
```

poster: 'https://m.media-amazon.com/images/M/MV5BMTU3NjE5NzYtYTYyNS00MDVmLWIwYjgtMmYwYWIxZDYyNzU2XkEyXkFqcGdeQXVyNzQzNzQxNzI@._V1_SY1000_SX677_AL_.jpg',
title: 'The Great Train Robbery',

fullplot: "Among the earliest existing films in American cinema - notable as the first film that presented a narrative story to tell - it depicts a group of cowboy outlaws who hold up a train and rob the passengers. They are then pursued by a Sheriff's posse. Several scenes have color included - all hand tinted.",

languages: ['English'],
released: ISODate("1903-12-01T00:00:00.000Z"),
directors: ['Edwin S. Porter'],
rated: 'TV-G',
awards: { wins: 1, nominations: 0, text: '1 win.' },
lastupdated: '2015-08-13 00:27:59.177000000',
year: 1903,
imdb: { rating: 7.4, votes: 9847, id: 439 },
countries: ['USA'],
type: 'movie',
tomatoes: {
viewer: { rating: 3.7, numReviews: 2559, meter: 75 },
fresh: 6,
critic: { rating: 7.6, numReviews: 6, meter: 100 },
rotten: 0,
lastUpdated: ISODate("2015-08-08T19:16:10.000Z")
}

1.Find all movies with full information from the 'movies' collection that released in the year 1893.

```
db.movies.find(
    { year : 1893 }
)
```

2. Find all movies with full information from the 'movies' collection that have a runtime greater than 120 minutes.

```
db.movies.find(
    { runtime : { $gt : 120 } }
)
```

3. Find all movies with full information from the 'movies' collection that have "Short" genre.

```
db.movies.find(
    {genres: "short"}
)
```

4. Retrieve all movies from the 'movies' collection that were directed by "William K.L. Dickson" and include complete information for each movie.

```
db.movies.find(
    {directors: "William K.L. Dickson"}
)
```

6. Retrieve all movies from the 'movies' collection that were released in the USA and include complete information for each movie.

```
db.movies.find(
    {countries : "USA"}
)
```

7. Retrieve all movies from the 'movies' collection that have complete information and are rated as "UNRATED".

```
db.movies.find(
    {rated : "UNRATED"}
)
```

8. Retrieve all movies from the 'movies' collection that have complete information and have received more than 1000 votes on IMDb.

```
db. movies .find(
    { "imdb. votes": { $gt : 100033
)
```

9. Retrieve all movies from the 'movies' collection that have complete information and have an IMDb rating higher than 7.

```
db. movies .find(
    { "imdb. rating": { $gt : 733
)
```

10. Retrieve all movies from the 'movies' collection that have complete information and have a viewer rating higher than 4 on Tomatoes.

```
db. movies. find(
    { "tomatoes. viewer. rating": { $gt : 433
)
```

11. Retrieve all movies from the 'movies' collection that have received an award.

```
db. movies. find(
    { "awards.wins": { $gt : 033
)
```

12. Find all movies with title, languages, released, directors, writers, awards, year, genres, runtime, cast, countries from the 'movies' collection in MongoDB that have at least one nomination.

```
db. movies .find(
    { "awards. nominations": { $gt : 033,
{ _id : 0,                  director : 1,      runtime : 1,
  title : 1,                writers : 1,       cast : 1,
  language : 1,             awards : 1,        countries : 1
  released : 1,             year : 1,          }
                            genres : 1,        )
```

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"
    },
    { _id:0,   languages:1,   writers:1,   runtime:1,
      released:1,   awards:1,   cast:1,
      title:1,   directors:1,   year:1,   countries:1
      genres:1,
    })
```

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 : ISODate("1893-05-09T00:00:00.002")
  },
  { _id:0,   languages:1,   directors:1,   countries:1
    title:1,   released:1,   writers:1,
  })
```

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 : {$regex: /scene/i}
    },
    {
      _id:0,
      title:1,
      languages:1,
      released:1,
      directors:1,
      writers:1,
      countries:1})
```

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

3. Write a script to insert two rows into the DEPT table. Name your script lab12_3.sql. Be sure to use the sequence that you created for the ID column. Add two departments named Education and
Administration. Confirm your additions. Run the commands in your script.
4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.

5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table.

1. CREATE SEQUENCE DEPT_ID_SEQ
START WITH 200
INCREMENT BY 10
MAXVALUE 1000
NOCYCLE;

2. SELECT sequence_name,
max_value,
increment_by,
last_number
FROM user_sequences
WHERE sequence_name = 'DEPT_ID_SEQ';

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	3
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

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, 'Administration');
SELECT * FROM DEPT;

4. CREATE INDEX EMP_DEPT_IDX
ON EMP(DEPT_ID);

5. SELECT index_name,
uniqueness,
column_name
FROM user_ind_columns
JOIN user_indexes USING (index_name)
WHERE table_name = 'EMP';

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?

Privilege : CREATE SESSION ; Type : system privilege.

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

CREATE TABLE

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

The owner of the table (or a DBA with special previleges)

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?

As a DBA, create a role with the required system previleges and then grant that role to all the users to simplify administration.

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. GRANT SELECT ON DEPARTMENTS TO other-user;

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.

Team1: INSERT INTO DEPARTMENTS (DEPTNO, DNAME) VALUES (500, 'Education');
SELECT * FROM team2. DEPARTMENTS;

Team 2: INSERT INTO DEPARTMENTS (DEPTNO, DNAME) VALUES (510, 'Human Resources');
SELECT * FROM team1. DEPARTMENTS;

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

SELECT * FROM USER_TABLES;

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

REVOKE SELECT ON DEPARTMENTS FROM other-users;

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

DELETE FROM DEPARTMENTS

WHERE DEPTNO = 500;

COMMIT;