RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

CS23332 DATABASE MANAGEMENT SYSTEMS LAB

Database Management Systems

Laboratory Record Note Book

Name : Shivani R J……. ……...

Year / Branch / Section : 2/CSE….. …….

University Register No. : 2116240701500…….. ……

College Roll No. : 240701500……..……

Semester : 3……..……

Academic Year : 2025-2026…….

RAJALAKSHMI
ENGINEERING COLLEGE

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

BONAFIDE CERTIFICATE

NAME .....

ACADEMIC YEAR ..... SEMESTER .....BRANCH.....

UNIVERSITY REGISTER No. [ ]

Certified that this is the bonafide record of work done by the above student in the

..... *Laboratory during the year* 20 - 20

Signature of Faculty - in - Charge

Submitted for the Practical Examination held on .....

Internal Examiner **External Examiner**

INDEX

Name : ____________________ **Branch :** ________ **Sec :** _______ **Roll No :** __________

S.No.	Date	Title	Page No.	Teacher's Sign / Remarks

Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables.

Terminologies Used in a Relational Database

1. A single ROW or table representing all data required for a particular employee. Each row should be identified by a primary key which allows no duplicate rows.

2. A COLUMN or attribute containing the employee number which identifies a unique employee. Here Employee number is designated as a primary key ,must contain a value and must be unique.

3. A column may contain foreign key. Here Dept_ID is a foreign key in employee table and it is a primary key in Department table.

4. A Field can be found at the intersection of a row and column. There can be only one value in it. Also it may have no value. This is called a null value.

EMP ID	FIRST NAME	LAST NAME	EMAIL
100	King	Steven	Sking
101	John	Smith	Jsmith
102	Neena	Bai	Neenba
103	Eex	De Haan	Ldehaan

Relational Database Properties

A relational database :

- Can be accessed and modified by executing structured query language (SQL) statements. ● Contains a collection of tables with no physical pointers.
- Uses a set of operators

Relational Database Management Systems

RDBMS refers to a relational database plus supporting software for managing users and processing SQL queries, performing backups/restores and associated tasks.
(Relational Database Management System) Software for storing data using SQL (structured query language). A relational database uses SQL to store data in a series of tables that not only record existing relationships between data items, but which also permit the data to be joined in new relationships. SQL (pronounced 'sequel') is based on a system of algebra developed by E F Codd, an IBM scientist who first defined the relational model in 1970. Relational databases are optimized for storing transactional data, and the majority of modern business software applications therefore use an RDBMS as their data store. The leading RDBMS vendors are Oracle, IBM and Microsoft.

The first commercial RDBMS was the Multics Relational Data Store, first sold in 1978. INGRES, Oracle, Sybase, Inc., Microsoft Access, and Microsoft SQL Server are wellknown database products and companies.Others include PostgreSQL, SQL/DS, and RDB. A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database. Most commercial RDBMS's use the Structured Query Language (SQL) to access the database, although SQL was invented after the development of the relational model and is not necessary for its use.
The leading RDBMS products are Oracle, IBM's DB2 and Microsoft's SQL Server. Despite repeated challenges by competing technologies, as well as the claim by some experts that no current RDBMS has fully implemented relational principles, the majority of new corporate databases are still being created and managed with an RDBMS.

SQL Statements

1. Data Retrieval(DR)
2. Data Manipulation Language(DML)
3. Data Definition Language(DDL)
4. Data Control Language(DCL)
5. Transaction Control Language(TCL)

TYPE	STATEMENT	DESCRIPTION
DR	SELECT	Retrieves the data from the database
DML	1.INSERT 3.DELETE 4.MERGE	Enter new rows, changes existing rows, 2.UPDATE removes unwanted rows from tables in the database respectively.
DDL	1.CREATE 2.ALTER 3.DROP 4.RENAME 5.TRUNCATE	Sets up, changes and removes data structures from tables.
TCL	1.COMMIT 2.ROLLBACK 3.SAVEPOINT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.
DCL	1.GRANT 2.RREVOKE	Gives or removes access rights to both the oracle database and the structures within it.

DATA TYPES

1. Character Data types:

- Char – fixed length character string that can varies between 1-2000 bytes

▪ Varchar / Varchar2 – variable length character string, size ranges from 1-4000 bytes.it saves the disk space(only length of the entered value will be assigned as the size of column) ▪ Long - variable length character string, maximum size is 2 GB

2. Number Data types : Can store +ve,-ve,zero,fixed point, floating point with 38 precission.

- Number – {p=38,s=0}
- Number(p) - fixed point
- Number(p,s) –floating point (p=1 to 38,s= -84 to 127)

3. Date Time Data type: used to store date and time in the table.

- DB uses its own format of storing in fixed length of 7 bytes for century, date, month, year, hour, minutes, and seconds.
- Default data type is “dd-mon-yy”
- New Date time data types have been introduced. They are

TIMESTAMP-Date with fractional seconds
INTERVAL YEAR TO MONTH-stored as an interval of years and months

INTERVAL DAY TO SECOND-stored as o interval of days to hour’s minutes and seconds

4. Raw Data type: used to store byte oriented data like binary data and byte string.

5. Other :

- CLOB – stores character object with single byte character.
- BLOB – stores large binary objects such as graphics, video, sounds.
- BFILE – stores file pointers to the LOB’s.

EXERCISE-1
Creating and Managing Tables

OBJECTIVE

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

- Create tables
- Describing the data types that can be used when specifying column definition
- Alter table definitions
- Drop, rename, and truncate tables

NAMING RULES

Table names and column names:

- Must begin with a letter
- Must be 1-30 characters long
- Must contain only A-Z, a-z, 0-9, _, $, and #
- Must not duplicate the name of another object owned by the same user ● Must not be an oracle server reserve words ● 2 different tables should not have same name.
- Should specify a unique column name.
- Should specify proper data type along with width
- Can include “not null” condition when needed. By default it is ‘null’.

The CREATE TABLE Statement

Table: Basic unit of storage; composed of rows and columns

Syntax: 1 Create table table_name (column_name1 data_ type (size)
column_name2 data_ type (size)….);

Syntax: 2 Create table table_name (column_name1 data_ type (size) 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));

Tables Used in this course

Creating a table by using a Sub query

SYNTAX
// CREATE TABLE table_name(column_name type(size)…);

Create table table_name **as** select column_name1,column_name2,……colmn_namen from
table_name where predicate;

AS Subquery
Subquery is the select statement that defines the set of rows to be inserted into the new table.

Example

Create table dept80 as select employee_id, last_name, salary*12 Annsal, hire_date
from employees where dept_id=80;

The ALTER TABLE Statement

The ALTER statement is used to
- Add a new column
- Modify an existing column
- Define a default value to the new column ● Drop a column
- To include or drop integrity constraint.

SYNTAX

ALTER TABLE table_name ADD /MODIFY(Column_name type(size));

ALTER TABLE table_name DROP COLUMN (Column_nname);

ALTER TABLE ADD CONSTRAINT Constraint_name PRIMARY KEY (Colum_Name); **Example:**

Alter table dept80 add (jod_id varchar2(9));
Alter table dept80 modify (last_name varchar2(30));
Alter table dept80 drop column job_id;

NOTE: Once the column is dropped it cannot be recovered.

DROPPING A TABLE

- All data and structure in the table is deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- Cannot roll back the drop table statement.

Syntax:

Drop table *tablename;*

Example:

Drop table dept80;

RENAMING A TABLE

To rename a table or view.
Syntax

RENAME old_name to new_name

Example:

Rename dept to detail_dept;

TRUNCATING A TABLE

Removes all rows from the table.

Releases the storage space used by that table.

Syntax
TRUNCATE TABLE *table_name*; E**xample:**

TRUNCATE TABLE copy_emp;

Find the Solution for the following:

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)

DEPARTMENT TABLE

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

JOB_GRADE TABLE

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

LOCATION TABLE

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

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

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

```
Create table dept(ID number(7),Name varchar2(25));
```

```
Table created.

0.03 seconds
```

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

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

```
create table em(ID number(7),first_name varchar2(25),last_name varchar2(25),dept_id
number(7));
```

```
Table created.

0.00 seconds
```

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

```
alter table em modify (last_name varchar2(50));
```

```
Table altered.

0.03 seconds
```

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 em2(ID number(7),first_name varchar2(25),last_name varchar2(25),Salary
number(20),dept_id number(7));
```

```
Results

Table created.

0.01 seconds
```

5. Drop the EMP table.

```
drop table em;
```

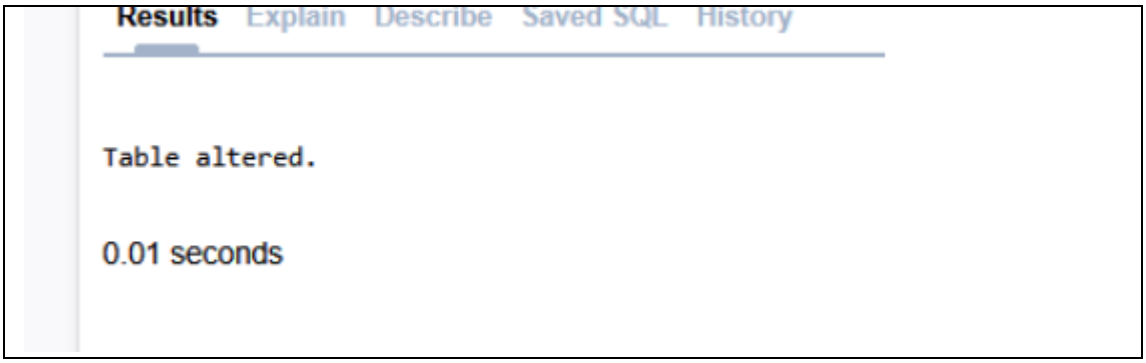
```
Results  Explain  Describe  Saved SQL  History

Table dropped.

0.05 seconds
```

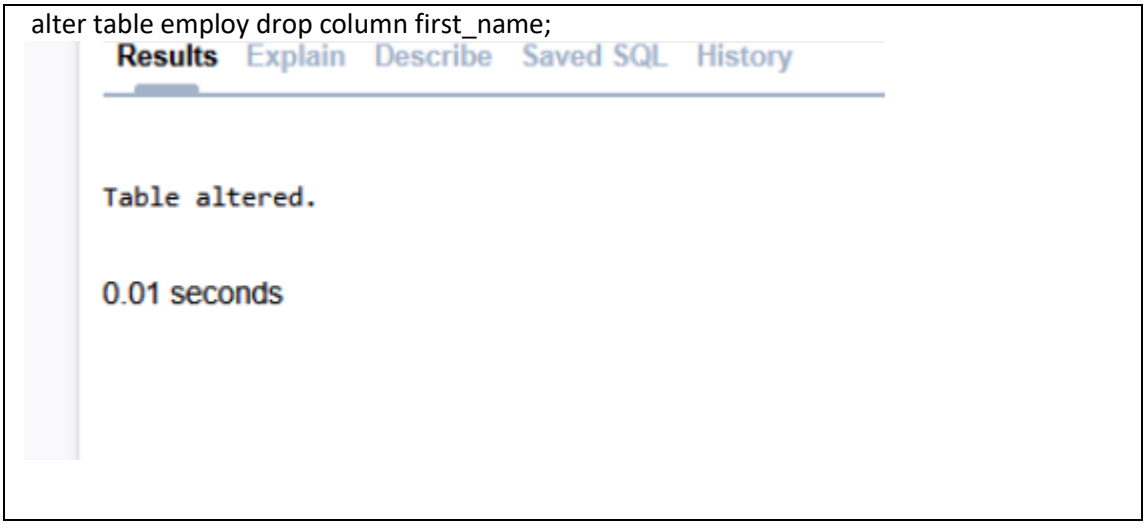
6. Rename the EMPLOYEES2 table as EMP.

```
alter table em2 rename to employ;
```



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

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



Evaluation Procedure	Marks awarded
Query(5)	

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

EXERCISE-2

MANIPULATING DATA

OBJECTIVE

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

- Describe each DML statement
- Insert rows into tables
- Update rows into table
- Delete rows from table
- Control Transactions

A DML statement is executed when you:

- Add new rows to a table
- Modify existing rows
- Removing existing rows

A transaction consists of a collection of DML statements that form a logical unit of work.

To Add a New Row

INSERT Statement

Syntax

INSERT INTO table_name VALUES (column1 values, column2 values, …, columnn values);

Example:

INSERT INTO department (70, 'Public relations', 100,1700);

Inserting rows with null values

Implicit Method: (Omit the column)

INSERT INTO department VALUES (30,'purchasing');

Explicit Method: (Specify NULL keyword)

INSERT INTO department VALUES (100,'finance', NULL, NULL);

Inserting Special Values

Example:

Using SYSDATE

INSERT INTO employees VALUES (113,'louis', 'popp', 'lpopp','5151244567',**SYSDATE**, 'ac_account', 6900, NULL, 205, 100);

Inserting Specific Date Values Example:

INSERT INTO employees VALUES (114,'den', 'raphealy', 'drapheal', '5151274561', **TO_DATE('feb 3,1999','mon, dd ,yyyy'),** 'ac_account', 11000,100,30);

To Insert Multiple Rows

& is the placeholder for the variable value **Example:**

INSERT INTO department VALUES (&dept_id, &dept_name, &location);

Copying Rows from another table

- Using Subquery **Example:**

```
INSER INTO sales_reps(id, name, salary, commission_pct)
        SELECt employee_id, Last_name, salary, commission_pct
FROM employees
WHERE jod_id LIKE '%REP');
```

CHANGING DATA IN A TABLE

UPDATE Statement

Syntax1: (to update specific rows**)**

UPDATE table_name SET column=value WHERE condition;

Syntax 2: (To updae all rows)

UPDATE table_name SET column=value;

Updating columns with a subquery

```
UPDATE employees
SET job_id= (SELECT job_id
FROM employees
WHERE employee_id=205)
WHERE employee_id=114;
```

REMOVING A ROW FROM A TABLE

DELETE STATEMENT

Syntax

DELETE FROM table_name WHERE conditions;

Example:

DELETE FROM department WHERE dept_name=’finance’’;

Find the Solution for the following:

1. Create MY_EMPLOYEE table with the following structure

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

3. Display the table with values.

```
create table my_employee(ID number(4),last_name varchar(25),first_name
varchar(25),userid varchar(25),Salary number(9,2));
insert into
my_employee(ID,last_name,first_name,userid,Salary)values(1,'Patel','Ralph','rpatel',895);
insert into
my_employee(Id,last_name,first_name,userid,Salary)values(2,'Dancs','betty','bdancs',860);
select * from my_employee;
```

Results Explain Describe Saved SQL History

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	betty	bdancs	860

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_employe(Id,last_name,first_name,userid,Salary)values(3,'biri','ben',substr('ben',1,1)||substr('biri',1,4),1100);
insert into
my_employe(Id,last_name,first_name,userid,Salary)values(4,'newman','chad',substr('chad,1,1)||substr('newman',1,6),7
50);
```

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

4 rows returned in 0.00 seconds Download

5. Make the data additions permanent.

```
commit;
```

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

```
update my_employe set last_name='Drexler'where ID=3;
```

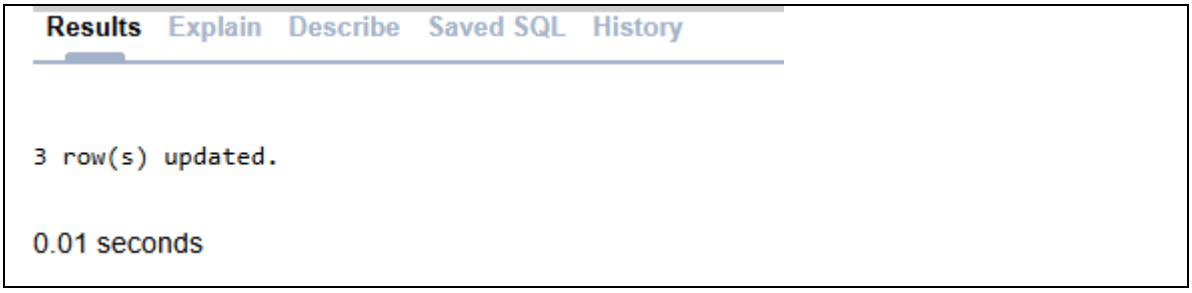
Results Explain Describe Saved SQL History

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

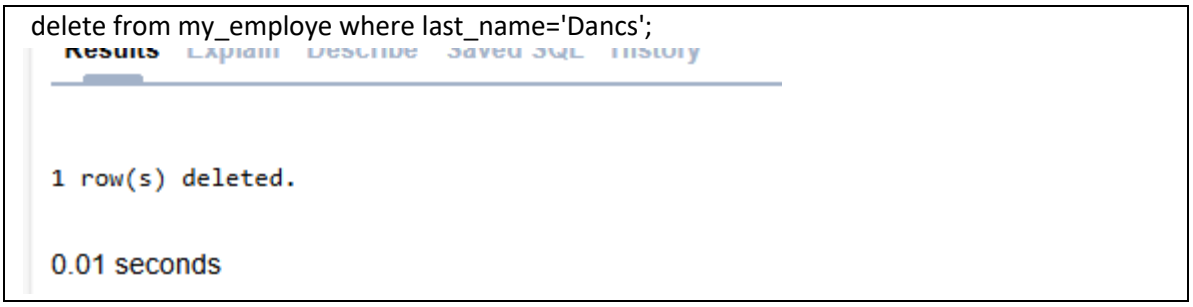
0.00 seconds

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

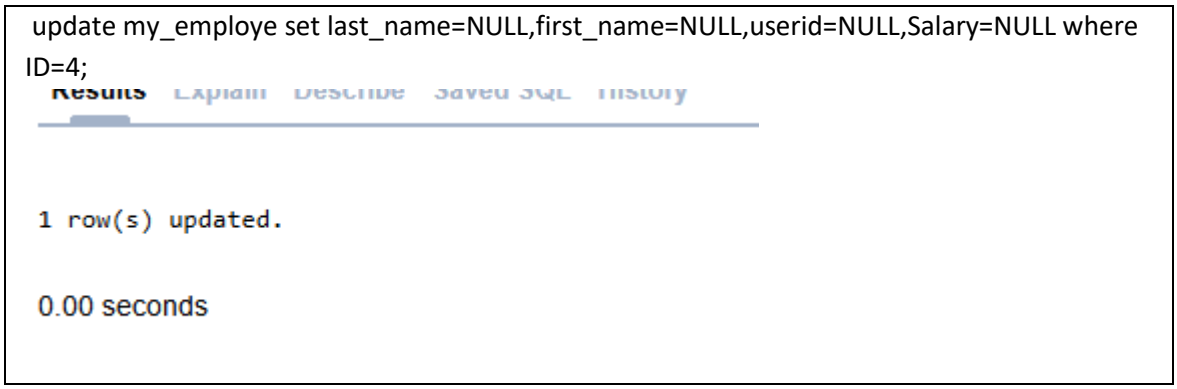
```
update my_employe set Salary=1000 where Salary<900;
```



8. Delete Betty dancs from MY _EMPLOYEE table.



9. Empty the fourth row of the emp table.



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

EXERCISE-3

INCLUDING CONSTRAINTS

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:

o Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful. o A null value is not equivalent to a value of zero. o A null value will always evaluate to null in any expression. o When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.

o 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';

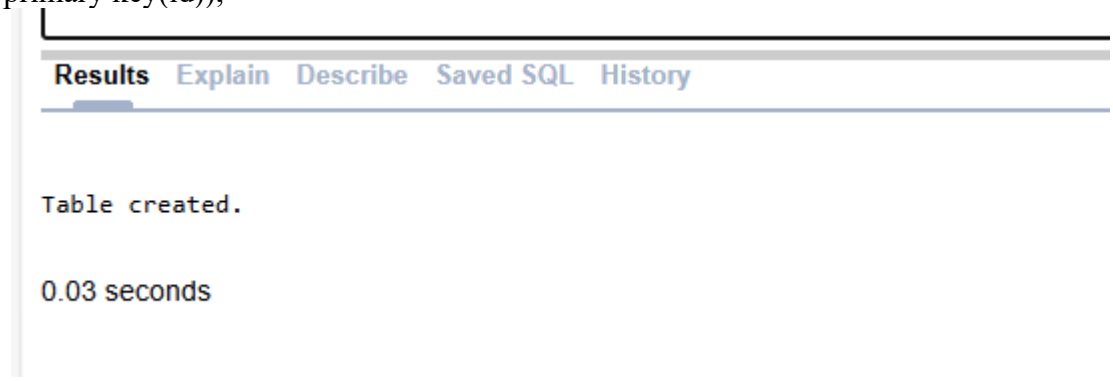
Viewing the columns associated with constraints

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 emp1(id number(5),first_name varchar2(20),last_name varchar2(10),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 depart(id number(10),dept_name varchar2(20),constraint my_dept_id_pk primary key(id));
```

Results Explain Describe Saved SQL History

```
Table created.
```

0.01 seconds

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 emp1 add dept_id number(25);
alter table emp1 add constraint my_emp_dept_id_fk foreign key(dept_id) references depart(id);
```

```
Table altered.
```

0.01 seconds

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 emp1 add commission number(2,2);
alter table emp1 add constraint commission_check check(commission>0);
```

Results Explain Describe Saved SQL History

```
Table altered.
```

0.01 seconds

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

EXERCISE-4

Writing Basic SQL SELECT Statements

OBJECTIVES

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

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

Capabilities of SQL SELECT statement

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

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

Basic SELECT Statement

Syntax

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

NOTE:

DISTINCT—Suppr ess
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;
Queries
select employee_id,last_name,sal*12 as "annual salary" from employees;

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

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
101	swetha	ragu	swe@gmail.com	231	08/05/2025	100	245	.45	467	243
102	aparna	kumar	apar@gmail.com	221	05/04/2025	103	249	.95	967	943
102	aparna	kumar	apar@gmail.com	221	05/04/2025	103	249	.95	967	943
103	mukunth	suresh	mukunth@gmail.com	265	06/04/2025	173	299	.85	767	983

4 rows returned in 0.00 seconds Download

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;

EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE
101	ragu	100	08/05/2025
102	kumar	103	05/04/2025
102	kumar	103	05/04/2025
103	suresh	173	06/04/2025

4 rows returned in 0.00 seconds Download

4. Provide an alias STARTDATE for the hire date.

select hire_date as STARTDATE from employees;

STARTDATE
08/05/2025
05/04/2025
05/04/2025
06/04/2025

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

select distinct job_id from employees;

JOB_ID
100
103
173

3 rows returned in 0.00 seconds

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;
```

employee and title
ragu,100
kumar,103
kumar,103
suresh,173

4 rows returned in 0.01 second

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

```
select
employee_id||','||first_name||','||last_name||','||email||','||phone_number||','||hire_date||','||job_id||','||salary||','||commiss
ion_pct||','||manager_id||','||department_id as "the output" from employees;
```

Results Explain Describe Saved SQL History

the output
101,swetha,ragu,swe@gmail.com,231,08/05/2025,100,245,.45,467,243
102,aparna,kumar,apar@gmail.com,221,05/04/2025,103,249,.95,967,943
102,aparna,kumar,apar@gmail.com,221,05/04/2025,103,249,.95,967,943
103,mukunth,suresh,mukunth@gmail.com,265,06/04/2025,173,299,.85,767,983

4 rows returned in 0.00 seconds Download

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

EXERCISE-5

Restricting and Sorting data

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

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

Limiting the Rows selected

- Using WHERE clause
- Alias cannot used in WHERE clause

Syntax

SELECT----------
FROM---------- WHERE
condition; **Example:**

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

Character strings and Dates

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

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

Example:
SELECT 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 employee where salary>12000;
```

Results Explain Describe Saved SQL History

LAST_NAME	SALARY
smith	15000

1 rows returned in 0.03 seconds Download

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

select last_name,department_id from employee where employee_id=176;

Results Explain Describe Saved SQL History

LAST_NAME	DEPARTMENT_ID
brown	30

1 rows returned in 0.00 seconds Download

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 employee where salary not between 5000 and 12000;

LAST_NAME	SALARY
smith	15000
taylor	4000

2 rows returned in 0.01 seconds

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

select last_name,job_id,hire_date from employee where hire_date between 02/20/1998 and 05/01/1998 order by hire_date asc;

LAST_NAME	JOB_ID	HIRE_DATE
davis	sa_man	02/25/1998
smith	mgr	03/15/1998

2 rows returned in 0.00 seconds Downloa

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 employee where department_id in(20,50) order by last_name asc;

LAST_NAME	DEPARTMENT_ID
allen	20
anderson	50
brownles	20
davis	20
green	50
king	50
smith	20
taylor	50

8 rows returned in 0.01 seconds Do

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 employee where salary between 5000 and 12000 and department_id in(20,50) order by last_name asc;

employee	monthly salary
allen	6000
anderson	8500
brownles	5000
davis	8000
green	11000
king	9500

6 rows returned in 0.00 seconds

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

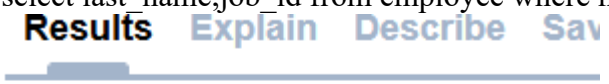
select last_name,hire_date from employee where hire_date like '%1994';

LAST_NAME	HIRE_DATE
king	03/01/1994
white	09/15/1994

2 rows returned in 0.01 seconds

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 employee where manager_id is null;



LAST_NAME	JOB_ID
hall	hr_rep
white	it_prog

2 rows returned in 0.01 seconds

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,commision_pct from employee where commision_pct is not null order by salary,commision_pct desc;

Results Explain Describe Saved SQL History

LAST_NAME	SALARY	COMMISION_PCT
king	9500	.2

1 rows returned in 0.00 seconds Download

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

```
select last_name from employee 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 employee where last_name like '%a%%e%';
```

LAST_NAME
allen
anderson

2 rows returned in 0.00 seconds

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 employee where job_id='sarep' or job_id='st_clerk' and salary not in(2500,3500,7000);
```

LAST_NAME	JOB_ID	SALARY
brownles	st_clerk	5000

1 rows returned in 0.01 seconds Do

13. Display the last name, salary, and commission for all employees whose commission amount is 20%.(hints:use predicate logic)

```
select last_name,salary,commision_pct from employee where commision_pct=0.2;
```

Results Explain Describe Saved SQL History

LAST_NAME	SALARY	COMMISION_PCT
king	9500	.2

1 rows returned in 0.00 seconds Download

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

EXERCISE-6

Single Row Functions

Objective

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

- Describe various types of functions available in SQL.
- Use character, number and date functions in SELECT statement.
- Describe the use of conversion functions.

Evaluation Procedure	Marks awarded
Practice Evaluation (5)	
Viva(5)	
Total (10)	
Faculty Signature	

Single row functions:

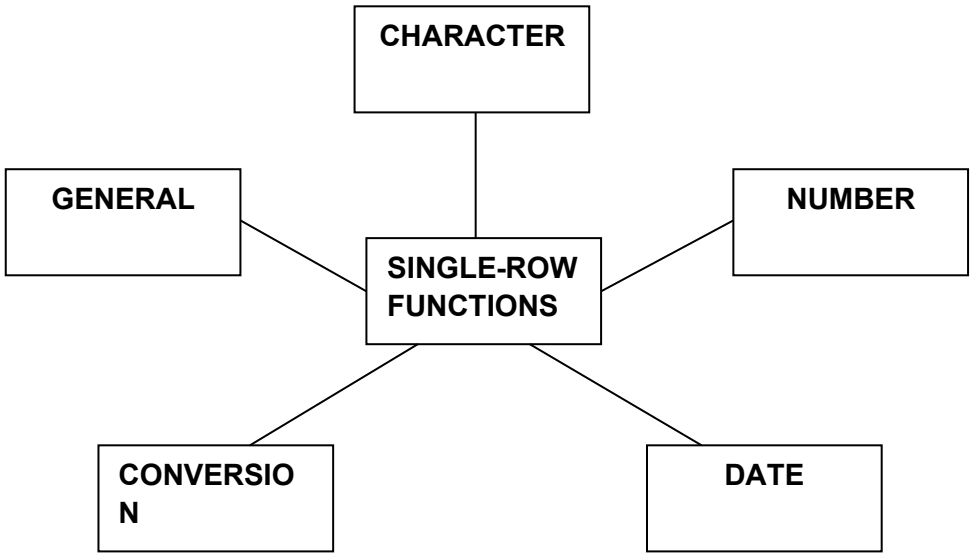
Manipulate data items.
Accept arguments and return one value.
Act on each row returned.
Return one result per row.
May modify the data type.
Can be nested.
Accept arguments which can be a column or an expression

Syntax

Function_name(arg1,…argn)

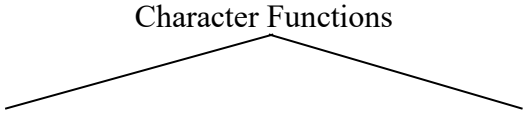
An argument can be one of the following

- ✓ User-supplied constant
- ✓ Variable value
- ✓ Column name
- ✓ Expression



- Character Functions: Accept character input and can return both character and number values.
- Number functions: Accept numeric input and return numeric values.
- Date Functions: Operate on values of the DATE data type.
- Conversion Functions: Convert a value from one type to another.

Character Functions



Case-manipulation functions

1. Lower
2. Upper
3. Initcap
4. Instr
5. Lpad/Rpad
6. Trim
7. Repalce

Character-manipulation functions

1. Concat
2. Substr
3. Length

Function	**Purpose**
lower(column/expr)	Converts alpha character values to lowercase
upper(column/expr)	Converts alpha character values to uppercase
initcap(column/expr)	Converts alpha character values the to uppercase for the first letter of each word, all other letters in lowercase
concat(column1/expr1, column2/expr2)	Concatenates the first character to the second character
substr(column/expr,m,n)	Returns specified characters from character value starting at character position m, n characters long
length(column/expr)	Returns the number of characters in the expression
instr(column/expr,'string',m,n)	Returns the numeric position of a named string

lpad(column/expr, n,'string')	Pads the character value right-justified to a total width of n character positions
rpad(column/expr,'string',m,n)	Pads the character value left-justified to a total width of n character positions
trim(leading/trailing/both, trim_character FROM trim_source)	Enables you to trim heading or string. trailing or both from a character
replace(text, search_string, replacement_string)	

Example:

lower('SQL Course')⇨sql course upper('SQL Course')⇨SQL COURSE
initcap('SQL Course')⇨Sql Course

SELECT 'The job id for'|| upper(last_name||'is'||lower(job_id) AS "EMPLOYEE DETAILS" FROM employees;

SELECT employee_id, last_name, department_id FROM employees
WHERE LOWER(last_name)='higgins';

Function	**Result**
CONCAT('hello', 'world')	helloworld
Substr('helloworld',1,5)	Hello
Length('helloworld')	10
Instr('helloworld','w')	6
Lpad(salary,10,'*')	*****24000
Rpad(salary,10,'*')	24000*****
Trim('h' FROM 'helloworld')	elloworld

Command	**Query**	**Output**
initcap(char);	*select initcap("hello") from dual;*	Hello
lower (char); upper (char);	*select lower ('HELLO') from dual;* *select upper ('hello') from dual;*	Hello HELLO
ltrim (char,[set]);	*select ltrim ('cseit', 'cse') from dual;*	IT
rtrim (char,[set]);	*select rtrim ('cseit', 'it') from dual;*	CSE
replace (char,search string, replace string);	*select replace ('jack and jue', 'j', 'bl') from dual;*	black and blue

substr (char,m,n);	*select substr ('information', 3, 4) from dual;*	form

Example:

SELECT employee_id, CONCAT (first_name,last_name) NAME , job_id,LENGTH(last_name), INSTR(last_name,'a') "contains'a'?"
FROM employees WHERE SUBSTR(job_id,4)='ERP';

NUMBER FUNCTIONS

Function	**Purpose**
round(column/expr, n)	Rounds the value to specified decimal
trunc(column/expr,n)	Truncates value to specified decimal
mod(m,n)	Returns remainder of division

Example

Function	Result
round(45.926,2)	45.93
trunc(45.926,2)	45.92
mod(1600,300)	100

SELECT ROUND(45.923,2), ROUND(45.923,0), ROUND(45.923,-1) FROM dual;

NOTE: Dual is a dummy table you can use to view results from functions and calculations.

SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-2) FROM dual;

SELECT last_name,salary,MOD(salary,5000) FROM employees WHERE job_id='sa_rep'; **Working with Dates**

The Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.

• The default date display format is DD-MON-RR.

– Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year

– Enables you to store 20th-century dates in the 21st century in the same way

Example

SELECT last_name, hire_date FROM employees WHERE hire_date < '01-FEB-88;

Working with Dates

SYSDATE is a function that returns:

• Date

• Time

Example

Display the current date using the DUAL table.

SELECT SYSDATE FROM DUAL;

Arithmetic with Dates

• Add or subtract a number to or from a date for a resultant date value.
• Subtract two dates to find the number of days between those dates.
• Add hours to a date by dividing the number of hours by 24.

Arithmetic with Dates

Because the database stores dates as numbers, you can perform calculations using arithmetic Operators such as addition and subtraction. You can add and subtract number constants as well as dates.
You can perform the following operations:

Operation	Result	Description
date + number	Date	Adds a number of days to a date
date – number	Date	Subtracts a number of days from a date
date – date	Number of days	Subtracts one date from another
date + number/24	Date	Adds a number of hours to a date

Example

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

Date Functions

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

Date Functions

Date functions operate on Oracle dates. All date functions return a value of DATE data type except MONTHS_BETWEEN, which returns a numeric value.

• MONTHS_BETWEEN(date1, date2)::: Finds the number of months between date1 and date2. The result can be positive or negative. If date1 is later than date2, the result is positive; if date1 is earlier than date2, the result is negative. The noninteger part of the result represents a portion of the month.

• ADD_MONTHS(date, n)::: Adds n number of calendar months to date. The value of n must be an integer and can be negative.

• NEXT_DAY(date, 'char')::: Finds the date of the next specified day of the week ('char') following date. The value of char may be a number representing a day or a character string.

• LAST_DAY(date)::: Finds the date of the last day of the month that contains date

• ROUND(date[,'fmt']):::: Returns date rounded to the unit that is specified by the format model fmt. If the format model fmt is omitted, date is rounded to the nearest day.

• TRUNC(date[, 'fmt']):::: Returns date with the time portion of the day truncated to the unit that is specified by the format model fmt. If the format model fmt is omitted, date is truncated to the nearest day.

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

Example

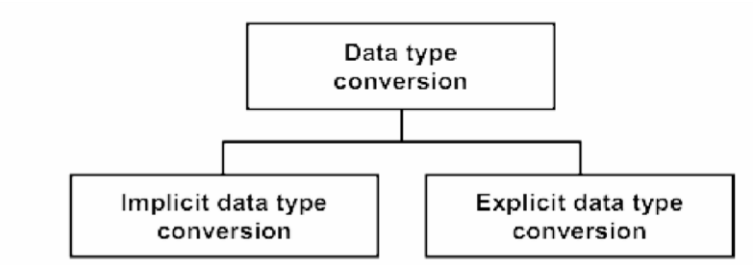
Display the employee number, hire date, number of months employed, sixmonth review date, first Friday after hire date, and last day of the hire month for all employees who have been employed for fewer than 70 months.

```
SELECT employee_id, hire_date,MONTHS_BETWEEN (SYSDATE, hire_date)
TENURE,ADD_MONTHS (hire_date, 6) REVIEW,NEXT_DAY (hire_date, 'FRIDAY'),
LAST_DAY(hire_date)
FROM   employees
WHERE  MONTHS_BETWEEN (SYSDATE, hire_date) < 70;
```

Conversion Functions

This covers the following topics:

• Writing a query that displays the current date
• Creating queries that require the use of numeric, character, and date functions
• Performing calculations of years and months of service for an employee



Implicit Data Type Conversion

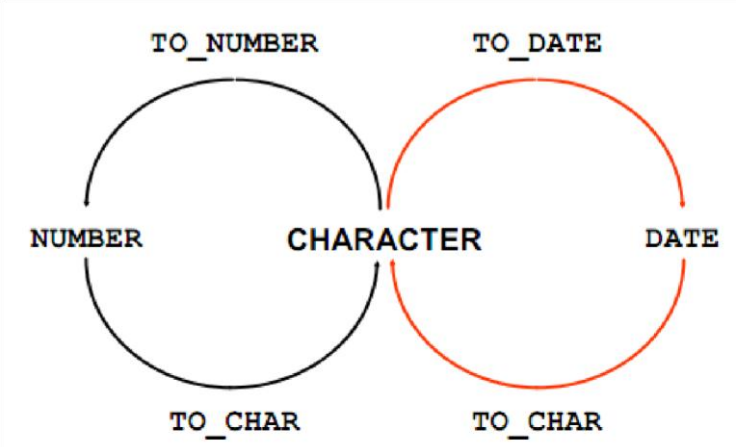
For assignments, the Oracle server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

For example, the expression hire_date > '01-JAN-90' results in the implicit conversion from the string '01-JAN-90' to a date.
For expression evaluation, the Oracle Server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

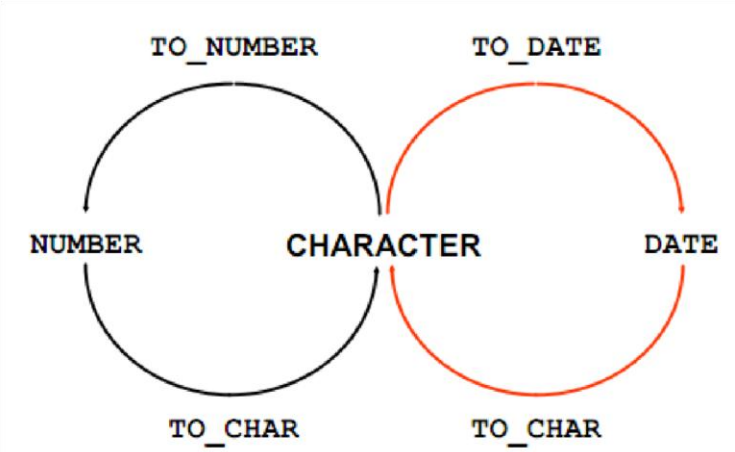
Explicit Data Type Conversion



SQL provides three functions to convert a value from one data type to another:
Example:
Using the TO_CHAR Function with Dates
TO_CHAR(date, 'format_model') **The format model:**
• Must be enclosed by single quotation marks
• Is case-sensitive
• Can include any valid date format element
• Has an fm element to remove padded blanks or suppress leading zeros
• Is separated from the date value by a comma

SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM employees WHERE last_name = 'Higgins';

Elements of the Date Format Model



Sample Format Elements of Valid Date

Element	Description
SCC or CC	Century; server prefixes B.C. date with -
Years in dates YYYY or SYYYY	Year; server prefixes B.C. date with -
YYY or YY or Y	Last three, two, or one digits of year
Y,YYY	Year with comma in this position
IYYY, IYY, IY, I	Four-, three-, two-, or one-digit year based on the ISO standard
SYEAR or YEAR	Year spelled out; server prefixes B.C. date with -
BC or AD	Indicates B.C. or A.D. year
B.C. or A.D.	Indicates B.C. or A.D. year using periods
Q	Quarter of year
MM	Month: two-digit value
MONTH	Name of month padded with blanks to length of nine characters
MON	Name of month, three-letter abbreviation
RM	Roman numeral month
WW or W	Week of year or month
DDD or DD or D	Day of year, month, or week
DAY	Name of day padded with blanks to a length of nine characters
DY	Name of day; three-letter abbreviation
J	Julian day; the number of days since December 31, 4713 B.C.

Date Format Elements: Time Formats

Use the formats that are listed in the following tables to display time information and literals and to change numerals to spelled numbers.

Element	Description
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12 or HH24	Hour of day, or hour (1–12), or hour (0–23)
MI	Minute (0–59)
SS	Second (0–59)
SSSSS	Seconds past midnight (0–86399)

Other Formats

Element	Description
/ . ,	Punctuation is reproduced in the result.
"of the"	Quoted string is reproduced in the result.

Specifying Suffixes to Influence Number Display

Element	Description
TH	Ordinal number (for example, DDTH for 4TH)
SP	Spelled-out number (for example, DDSP for FOUR)
SPTH or THSP	Spelled-out ordinal numbers (for example, DDSPTH for FOURTH)

Example

```
SELECT last_name,
TO_CHAR(hire_date, 'fmDD Month YYYY') AS HIREDATE
FROM   employees;
```

Modify example to display the dates in a format that appears as "Seventeenth of June 1987 12:00:00 AM."

```
SELECT last_name,
TO_CHAR (hire_date, 'fmDdspth "of" Month YYYY fmHH:MI:SS AM') HIREDATE
FROM   employees;
```

Using the TO_CHAR Function with Numbers

TO_CHAR(number, 'format_model')

These are some of the format elements that you can use with the TO_CHAR function to display a number value as a character:

Element	Result
9	Represents a number
0	Forces a zero to be displayed
$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a comma as thousands indicator

Number Format Elements

If you are converting a number to the character data type, you can use the following format elements:

Element	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
$	Floating dollar sign	$999999	$1234
L	Floating local currency symbol	L999999	FF1234
D	Returns in the specified position the decimal character. The default is a period (.).	99D99	99.99
.	Decimal point in position specified	999999.99	1234.00
G	Returns the group separator in the specified position. You can specify multiple group separators in a number format model.	9,999	9G999
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation (format must specify four Es)	99.999EEEE	1.234E+03
U	Returns in the specified position the "Euro" (or other) dual currency	U9999	€1234
V	Multiply by 10 *n* times (*n* = number of 9s after V)	9999V99	123400
S	Returns the negative or positive value	S9999	-1234 or +1234
B	Display zero values as blank, not 0	B9999.99	1234.00

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

Using the TO_NUMBER and TO_DATE Functions

• Convert a character string to a number format using the TO_NUMBER function:
TO_NUMBER(char[, 'format_model']

• Convert a character string to a date format using the TO_DATE function:
TO_DATE(char[, 'format_model']

• These functions have an fx modifier. This modifier specifies the exact matching for the character argument and date format model of a TO_DATE function.

The fx modifier specifies exact matching for the character argument and date format model of a TO_DATE function:

• Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.

• The character argument cannot have extra blanks. Without fx, Oracle ignores extra blanks.

• Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without fx, numbers in the character argument can omit leading zeros.

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY');
```

Find the Solution for the following:

1. Write a query to display the current date. Label the column Date.

```
select sysdate as "Date" from dual;
```

Date
08/19/2025

1 rows returned in 0.00 seconds

2. The HR department needs a report to display the employee number, last name, salary, and increased by 15.5% (expressed as a whole number) for each employee. Label the column New Salary.

```
select emp_id,last_name,round(salary+salary*0.155) as "New salary" from emp1;
```

EMP_ID	LAST_NAME	New salary
101	higgins	5775
102	jones	7161
105	anderson	8316
109	jackson	5198
110	brown	3465
103	miller	4620
104	johnson	4043
106	martin	3234
107	smith	6468
108	taylor	9240

10 rows returned in 0.00 seconds

3. Modify your query lab_03_02.sql to add a column that subtracts the old salary from the new salary. Label the column Increase.

```
alter table emp1 add increase number(10,2);
update emp1 set increase= salary *0.155;
```

4. Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters J, A, or M. Give each column an appropriate label. Sort the results by the employees' last names.

select initcap(last_name) as "Last Name",length(last_name) as "Name length" from emp1 where upper(substr(last_name,1,1)) in ('J','A','M') order by last_name;

Last Name	Name length
Anderson	8
Jackson	7
Johnson	7
Jones	5
Martin	6
Miller	6

6 rows returned in 0.01 seconds

5. Rewrite the query so that the user is prompted to enter a letter that starts the last name. For example, if the user enters H when prompted for a letter, then the output should show all employees whose last name starts with the letter H.

initcap(last_name) as "Last Name",length(last_name) as "Name length" from emp1 where upper(substr(last_name,1,1))=upper('&letter') order by last_name;

6. The HR department wants to find the length of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

select last_name,round(months_between(sysdate,hire_date)) as "MONTHS_WORKED" from emp1 order by MONTHS_WORKED;

LAST_NAME	MONTHS_WORKED
brown	42
smith	51
martin	60
anderson	77
johnson	85
taylor	92
jackson	112
miller	118
jones	163
higgins	182

10 rows returned in 0.00 seconds D

Note: Your results will differ.

7. Create a report that produces the following for each employee:
<employee last name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries.

```
select last_name||'earns'||salary||'monthly but wants'||(salary*3) as "Dream salaries" from emp1;
```

Dream salaries
higginsearns5000monthly but wants15000
jonesearns6200monthly but wants18600
andersonearns7200monthly but wants21600
jacksonearns4500monthly but wants13500

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the $ symbol. Label the column SALARY.

select last_name,lpad(salary,15,'$') as "SALARY" from emp1;

LAST_NAME	SALARY
higgins	$$$$$$$$$$5000
jones	$$$$$$$$$$6200
anderson	$$$$$$$$$$7200
jackson	$$$$$$$$$$4500
brown	$$$$$$$$$$3000
miller	$$$$$$$$$$4000
johnson	$$$$$$$$$$3500
martin	$$$$$$$$$$2800
smith	$$$$$$$$$$5600
taylor	$$$$$$$$$$8000

9. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."
select last_name,to_char(hire_date,'Day,"the" ddspth "of" month,YYYY')as "Hire Date",to_char(next_day(add_months(hire_date,6),'Monday'),'Day,"the",Ddspth "of" Month,YYYY')as REVIEW from emp1;

LAST_NAME	Hire Date	REVIEW
higgins	Thursday ,the seventeenth of june ,2010	Monday ,the,Twentieth of December ,2010
jones	Wednesday,the twenty-fifth of january ,2012	Monday ,the,Thirtieth of July ,2012
anderson	Tuesday ,the nineteenth of march ,2019	Monday ,the,Twenty-Third of September,2019
jackson	Saturday ,the ninth of april ,2016	Monday ,the,Tenth of October ,2016
brown	Monday ,the fourteenth of february ,2022	Monday ,the,Fifteenth of August ,2022
miller	Tuesday ,the third of november ,2015	Monday ,the,Ninth of May ,2016
johnson	Sunday ,the fifteenth of july ,2018	Monday ,the,Twenty-First of January ,2019
martin	Tuesday ,the first of september,2020	Monday ,the,Eighth of March ,2021
smith	Thursday ,the twentieth of may ,2021	Monday ,the,Twenty-Second of November ,2021

10. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.

select last_name,hire_date,to_char(hire_date,'Day') as"DAY" from emp1 order by to_char(hire_date,'D');

LAST_NAME	HIRE_DATE	DAY
johnson	07/15/2018	Sunday
brown	02/14/2022	Monday
taylor	12/11/2017	Monday
miller	11/03/2015	Tuesday
martin	09/01/2020	Tuesday
anderson	03/19/2019	Tuesday
jones	01/25/2012	Wednesday
smith	05/20/2021	Thursday
higgins	06/17/2010	Thursday
jackson	04/09/2016	Saturday

10 rows returned in 0.01 seconds Download

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	

Total (15)	
Faculty Signature	

EXERCISE-7

Displaying data from multiple tables

Objective

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

• Write SELECT statements to access data from more than one table using equality and nonequality joins
• View data that generally does not meet a join condition by using outer joins
• Join a table to itself by using a self join
Sometimes you need to use data from more than one table.

Cartesian Products

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

Example:

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

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

Types of Joins

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

Joining Tables Using Oracle Syntax

```
SELECT table1.column, table2.column
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 nonequijoin. 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 last_name,dept_id,dept_name from employee,dept where employee.department_id=dept.dept_id;

LAST_NAME	DEPT_ID	DEPT_NAME
Taylor	10	Accounting
Allen	30	Sales
Smith	80	IT
Adams	80	IT

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

select distinct dept_name,dept.location_id from dept,loction where dept.dept_id=80 and dept.location_id=loction.location_id;

DEPT_NAME	LOCATION_ID
IT	4

1 rows returned in 0.02 seconds Download

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.dept_name,d.location_id,l.city from employee e join dept d on e.department_id=d.dept_id join loction l on d.location_id=l.location_id where e.commission_pct is not null;

LAST_NAME	DEPT_NAME	LOCATION_ID	CITY
Allen	Sales	3	Toronto
Adams	IT	4	Toronto

2 rows returned in 0.02 seconds Download

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

select last_name,dept_name from employee e,dept d where last_name like '%a%'and e.department_id=d.dept_id;

LAST_NAME	DEPT_NAME
Taylor	Accounting
Adams	IT

2 rows returned in 0.01 seconds Download

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

select last_name,job_id,dept_id,dept_name from employee e join dept d on e.department_id=d.dept_id join loction l on d.location_id=l.location_id where city='Toronto';

LAST_NAME	JOB_ID	DEPT_ID	DEPT_NAME
Smith	IT_PROG	80	IT
Adams	IT_PROG	80	IT
Miller	SA_REP	30	Sales
Lee	IT_PROG	80	IT

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 last_name,employee_id as emp#,dept.manager_id as mdr# from employee,dept;

LAST_NAME	EMP#	MDR#
Allen	103	1001
Clark	108	1001
Smith	101	1001

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

SELECT LAST_NAME,DEPT_NAME FROM employee E ,DEPT D WHERE E.MANAGER_ID IS NULL AND E.DEPARTMENT_ID=D.DEPT_ID ORDER BY EMPLOYEE_ID;

LAST_NAME	DEPT_NAME
King	Accounting

1 rows returned in 0.02 seconds Download

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, d.DEPT_NAME FROM employee e
JOIN dept d ON e.DEPARTMENT_ID = d.DEPT_ID
WHERE e.DEPARTMENT_ID = (
   SELECT DEPARTMENT_ID
   FROM employee
   WHERE FIRST_NAME = 'Jane' AND LAST_NAME = 'Adams'
);
```

LAST_NAME	DEPT_NAME
Smith	IT
Adams	IT
Lee	IT

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

```
SELECT LAST_NAME,JOB_ID,DEPT_NAME,SALARY,GRADE_LEVEL FROM employee E JOIN Dept D
ON E.DEPARTMENT_ID=D.DEPT_ID
JOIN JOB_GRADE J ON E.SALARY BETWEEN J.LOWEST_SAL AND J.HIGHEST_SAL;
```

LAST_NAME	JOB_ID	DEPT_NAME	SALARY	GRADE_LEVEL
Clark	AC_ACCOUNT	Accounting	5000	A
Taylor	AC_ACCOUNT	Accounting	7000	B
King	AD_PRES	Accounting	10000	C

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

```
SELECT LAST_NAME AS EMPLOYEE_NAME,HIRE_DATE
```

```
FROM employee
WHERE HIRE_DATE > (
   SELECT HIRE_DATE
   FROM employee
   WHERE LAST_NAME = 'Davies'
);
```

EMPLOYEE_NAME	HIRE_DATE
Smith	1/15/2020
Adams	7/10/2019
Miller	11/5/2019

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 employee e
JOIN employee m ON e.MANAGER_ID = m.EMPLOYEE_ID
WHERE e.HIRE_DATE < m.HIRE_DATE;
```

Employee	Emp Hired	Manager	Mgr Hired
Clark	1/10/2010	Taylor	8/30/2017

1 rows returned in 0.01 seconds Download

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	

Viva(5)	
Total (15)	
Faculty Signature	

EXERCISE-8

Aggregating Data Using Group Functions

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

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

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

The HR department needs the following reports:

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

select round(max(salary))as MAXIMUM,round(min(salary)) as MINIMUN,round(sum(salary)) as SUM,round(avg(salary)) as AVERAGE from employee;

MAXIMUM	MINIMUN	SUM	AVERAGE
10000	5000	58400	6489

rows returned in 0.02 seconds Download

5. Modify the above query to display the minimum, maximum, sum, and average salary for each job type.
select job_id, round(max(salary))as MAXIMUM,round(min(salary)) as MINIMUN,round(sum(salary)) as SUM,round(avg(salary)) as AVERAGE from employee group by job_id;

JOB_ID	MAXIMUM	MINIMUN	SUM	AVERAGE
AC_ACCOUNT	7000	5000	12000	6000
IT_PROG	6500	6000	25100	6275
AD_PRES	10000	10000	10000	10000
SA_REP	5800	5500	11300	5650

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

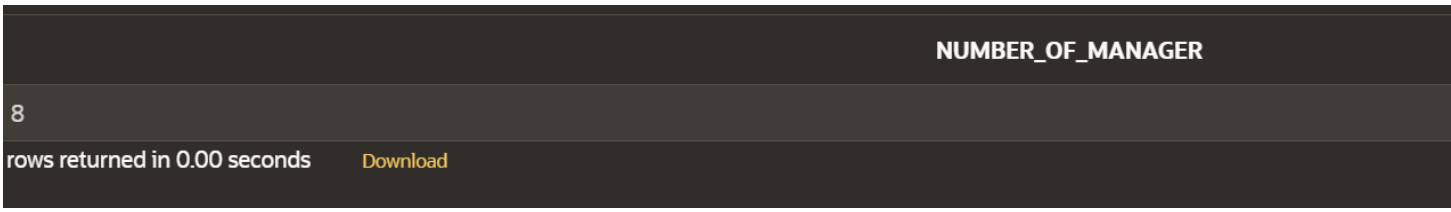
select job_id,count(*) as Number_of_people from employee where job_id=:job_title group by job_id;

JOB_ID	NUMBER_OF_PEOPLE
IT_PROG	4

rows returned in 0.01 seconds Download

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(manager_id) as number_of_manager from employee where manager_id is not null;

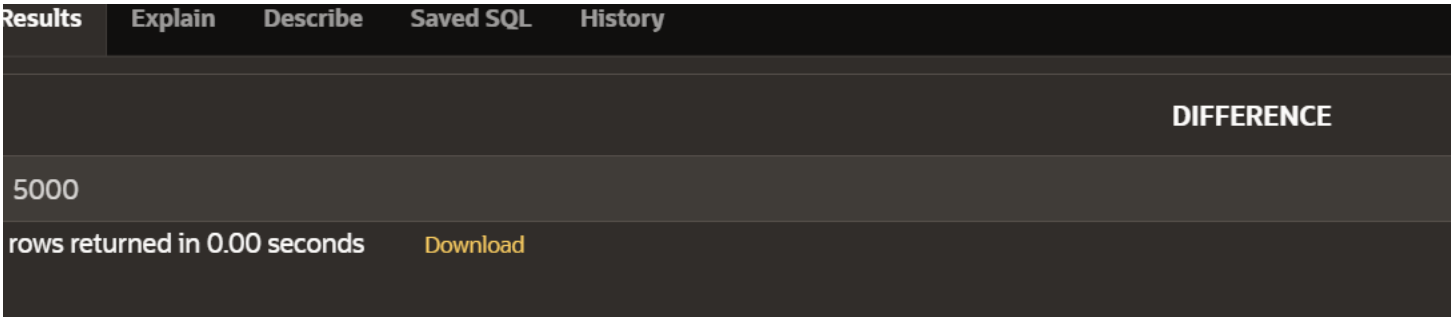


NUMBER_OF_MANAGER
8

rows returned in 0.00 seconds Download

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

select max(salary)-min(salary) as DIFFERENCE from employee;



Results Explain Describe Saved SQL History

DIFFERENCE
5000

rows returned in 0.00 seconds Download

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) from employee where manager_id is not null group by manager_id having min(salary)>6000 order by min(salary) desc;

MANAGER_ID	MIN(SALARY)
1001	7000

1 rows returned in 0.00 seconds Download

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 in 1995",
  SUM(CASE WHEN TO_CHAR(hire_date, 'YYYY') = '1996' THEN 1 ELSE 0 END) AS "Hired in 1996",
  SUM(CASE WHEN TO_CHAR(hire_date, 'YYYY') = '1997' THEN 1 ELSE 0 END) AS "Hired in 1997",
  SUM(CASE WHEN TO_CHAR(hire_date, 'YYYY') = '1998' THEN 1 ELSE 0 END) AS "Hired in 1998"
FROM employees;
```

Total Employees	Hired in 1995	Hired in 1996	Hired in 1997	Hired in 1998
9	0	0	0	0

rows returned in 0.01 seconds Download

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,department_id,sum(salary) from employee where department_id in (20,50,80,90) group by job_id,department_id order by department_id,job_id;
```

JOB_ID	DEPARTMENT_ID	SUM(SALARY)
IT_PROG	80	25100

rows returned in 0.04 seconds Download

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.dept_name as names,d.location_id as locations ,count(employee_id) as no_of_people,round(avg(e.salary),2)as salary from department d join employee e on d.dept_id=e.department_id group by d.dept_name,d.location_id order by d.dept_name;

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

EXERCISE-9

Sub queries

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 employee where department_id = (select department_id from employee where last_name=:last_name) and last_name not in ('adams');

LAST_NAME	HIRE_DATE
Smith	1/15/2020
Davies	5/1/2019
Lee	2/20/2021

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.

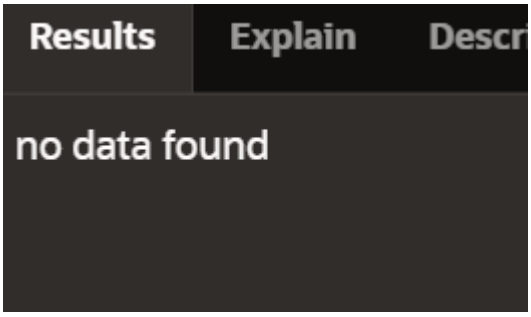
select employee_id,last_name,salary from employee where salary >(select avg(salary) from employee) order by salary asc;

EMPLOYEE_ID	LAST_NAME	SALARY
102	Adams	6500
104	Taylor	7000
105	King	10000

rows returned in 0.00 seconds Download

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 employee where department_id in (select last_name from employee where last_name like '%u%');

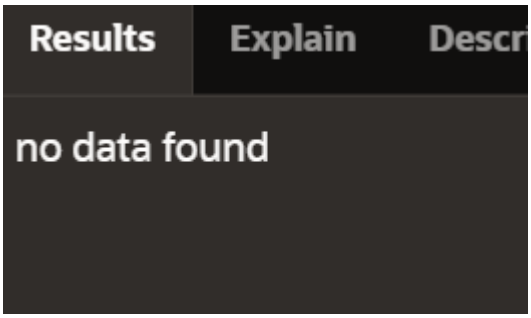


4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

select last_name,department_id,job_id from employee ,department where LOCATION_ID =1700 ;

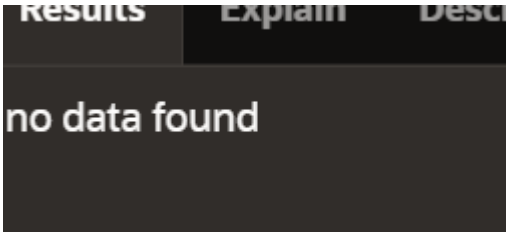
5. Create a report for HR that displays the last name and salary of every employee who reports to King.

elect last_name,salary from employee where manager_id=(select employee_id from employee where last_name='king');



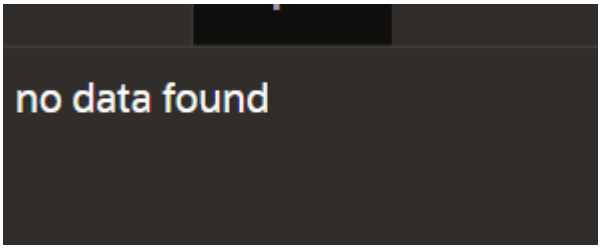
6. Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

select department_id,last_name,job_id from employee e join department d on e.department_id=d.dept_id where dept_name='IT';



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

select employee_id,last_name,salary from employee where salary>(select avg(salary) from employee) and department_id in (select distinct department_id from employee where last_name like '%u%');



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

EXERCISE-10

USING THE SET OPERATORS

Objectives

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

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

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

Queries containing set operators are called *compound queries*.

Operator	Returns
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query, including all duplicates
INTERSECT	All distinct rows selected by both queries
MINUS	All distinct rows that are selected by the first SELECT statement and not selected in the second SELECT statement

The tables used in this lesson are:

- EMPLOYEES: Provides details regarding all current employees

• JOB_HISTORY: Records the details of the start date and end date of the former job, and the job identification number and department when an employee switches jobs

UNION Operator

Guidelines

• The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.

• UNION operates over all of the columns being selected.

• NULL values are not ignored during duplicate checking.

• The IN operator has a higher precedence than the UNION operator.

• 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.

DEPT_ID
10
20
30
80

4 rows returned in 0.02 seconds Download

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.

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

JOB_ID	DEPARTMENT_ID
AC_ACCOUNT	10
AD_PRES	20

2 rows returned in 0.01 seconds Download

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).

EMPLOYEE_ID	JOB_ID
101	IT_PROG
105	AD_PRES

2 rows returned in 0.02 seconds Download

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.

NAME	DEPT_ID
Adams	80
Allen	30
Clark	10
Davies	80

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	

Total (15)	
Faculty Signature	

EXERCISE-11

CREATING VIEWS

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 (W**ithout 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.

```
View created.

0.03 seconds
```

2. Display the contents of the EMPLOYEES_VU view.

EMPLOYEE	DEPARTMENT	SALARY	GRADE
Smith	IT	6000	A
Smith	IT	6000	D
Smith	IT	6000	C
Smith	IT	6000	B
King	Research	10000	A

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

DEPT80	SELECT EMPLOYEE_ID,LAST_NAME,DEPARTMENT_ID FROM EMPP WHERE DEPARTMENT_ID=80 WITH CHECK OPTION
EMPLOYEE_VU	SELECT EMPLOYEE_ID,LAST_NAME,DEPARTMENT_ID FROM EMPP
EMPLOYEE_VU1	SELECT EMPLOYEE_ID,LAST_NAME,DEPARTMENT_ID FROM EMPP
SALARY_VU	SELECT LAST_NAME,DEPT_NAME,SALARY,GRADE_LEVEL FROM EMPP E JOIN DEPTT D ON E.DEPARTMENT_ID=D.DEPT_ID CROSS JOIN JOB_GRADE

4 rows returned in 0.13 seconds Download

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

EMPLOYEE	DEPT_NUM
Smith	80
King	20
Lee	80

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.

```
View created.

0.04 seconds
```

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

EMPNO	EMPLOYEE	DEPTNO
101	Smith	80
106	Lee	80
109	Davies	80

7. Attempt to reassign Matos to department 80.

```
0 row(s) updated.

0.06 seconds
```

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.

```
View created.

0.04 seconds
```

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

EXERCISE 12

Intro to Constraints; NOT NULL and UNIQUE Constraints

Global Fast Foods has been very successful this past year and has opened several new stores. They need to add a table to their database to store information about each of their store's locations. The owners want to make sure that all entries have an identification number, date opened, address, and

city and that no other entry in the table can have the same email address. Based on this information, answer the following questions about the global_locations table. Use the table for your answers.

Global Fast Foods global_locations Table						
NAME	**TYPE**	**LENGTH**	**PRECISION**	**SCALE**	**NULLABLE**	**DEFAULT**
Id						
name						
date_opened						
address						
city						
zip/postal code						
phone						
email						
manager_id						
Emergency contact						

1. What is a "constraint" as it relates to data integrity?

A database constraint is a rule applied to data within a table to ensure its accuracy, consistency, and reliability, upholding data integrity by preventing invalid entries or actions.

2. What are the limitations of constraints that may be applied at the column level and at the table level?

A database constraint is a rule applied to data within a table to ensure its accuracy, consistency, and reliability, upholding data integrity by preventing invalid entries or actions.

3. Why is it important to give meaningful names to constraints?

Meaningful names make constraints easier to identify, manage (e.g., enable/disable/drop), and understand their purpose, especially during troubleshooting or schema changes. They improve database maintainability and collaboration among developers.

4. Based on the information provided by the owners, choose a datatype for each column. Indicate the length, precision, and scale for each NUMBER datatype.

Id: INT PRIMARY KEY (ensures uniqueness and non-nullability for identification)

name: VARCHAR(255) NOT NULL (stores varying length strings for the store name, mandatory)

date opened: DATE NOT NULL (stores date values for when the store opened, mandatory)

address: VARCHAR(255) NOT NULL (stores varying length strings for the address, mandatory)

city: VARCHAR(100) NOT NULL (stores varying length strings for the city, mandatory)

zip/postal code: VARCHAR(10) NULLABLE (stores varying length strings for the code, optional)

phone: VARCHAR(20) NULLABLE (stores varying length strings for the phone, optional)

email: VARCHAR(255) UNIQUE NOT NULL (stores varying length strings for email, must be unique and mandatory)

manager id: INT NULLABLE (stores integer for manager ID, optional)

Emergency contact: VARCHAR(255) NULLABLE (stores varying length strings for emergency contact, optional)

5. Use “(nullable)” to indicate those columns that can have null values.

Id (NOT NULL as it's a PRIMARY KEY)

name (NOT NULL)

date opened (NOT NULL)

address (NOT NULL)

city (NOT NULL)

zip/postal code (nullable)

phone (nullable)

email (NOT NULL, UNIQUE)

manager id (nullable)

Emergency contact (nullable)

6. Write the CREATE TABLE statement for the Global Fast Foods locations table to define the constraints at the column level.

CREATE TABLE global_fast_foods_locations (id NUMBER PRIMARY KEY, loc_name VARCHAR2(20) NOT NULL, address VARCHAR2(30) NOT NULL, city VARCHAR2(20) NOT NULL, zip_postal VARCHAR2(20) NOT NULL, phone VARCHAR2(15), email VARCHAR2(80), manager_id NUMBER, contact VARCHAR2(40));

7. Execute the CREATE TABLE statement in Oracle Application Express.

Executed

8. Execute a DESCRIBE command to view the Table Summary information.

DESCRIBE global_fast_foods_locations;

9. Rewrite the CREATE TABLE statement for the Global Fast Foods locations table to define the UNIQUE constraints at the table level. Do not execute this statement.

CREATE TABLE global_fast_foods_locations (id NUMBER, loc_name VARCHAR2(20) NOT NULL, address VARCHAR2(30) NOT NULL, city VARCHAR2(20) NOT NULL, zip_postal VARCHAR2(20) NOT NULL, phone VARCHAR2(15), email VARCHAR2(80), manager_id NUMBER, contact VARCHAR2(40), CONSTRAINT pk_global_locations PRIMARY KEY (id), CONSTRAINT uk_loc_name UNIQUE (loc_name), CONSTRAINT uk_email UNIQUE (email));

NAME	TYPE	LENGTH	PRECISION	SCALE	NULLABLE	DEFAULT
id	number	4				
loc_name	varchar2	20			X	
	date					
address	varchar2	30				
city	varchar2	20				
zip_postal	varchar2	20			X	
phone	varchar2	15			X	
email	varchar2	80			X	
manager_id	number	4			X	
contact	varchar2	40			X	

Evaluation Procedure	Marks awarded

Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

EXERCISE 13
Creating Views

1. What are three uses for a view from a DBA's perspective?

A view can restrict access to specific rows and columns

2. Create a simple view called view_d_songs that contains the ID, title and artist from the DJs on Demand table for each "New Age" type code. In the subquery, use the alias "Song Title" for the title column.

```
CREATE VIEW view_d_songs AS
SELECT id, title AS "Song Title", artist
FROM DJs_on_Demand
WHERE type_code = 'New Age';
```

```
View VIEW_D_SONGS created.

Elapsed: 00:00:00.011
```

3. SELECT * FROM view_d_songs. What was returned?

```
CREATE OR REPLACE VIEW view_d_songs AS
SELECT id AS "Song ID", title AS "Song Title", artist AS "Song Artist", type_code
AS "Music Category"
FROM DJs_on_Demand
WHERE type_code = 'New Age'
```

```
View VIEW_D_SONGS created.

Elapsed: 00:00:00.015
```

4. REPLACE view_d_songs. Add type_code to the column list. Use aliases for all columns.

```
CREATE VIEW view_j_event_list AS
SELECT event_name AS "Event Name",
event_date AS "Event Date",
theme_description AS "Theme"
FROM Events;
```

Or use alias after the CREATE statement as shown.

5. Jason Tsang, the disk jockey for DJs on Demand, needs a list of the past events and those planned for the coming months so he can make arrangements for each event's equipment setup. As the company manager, you do not want him to have access to the price that clients paid for their events. Create a view for Jason to use that displays the name of the event, the event date, and the theme description. Use aliases for each column name.

```
CREATE VIEW view_j_event_list AS
SELECT event_name AS "Event Name",
event_date AS "Event Date",
theme_description AS "Theme"
FROM Events;
```

6. It is company policy that only upper-level management be allowed access to individual employee salaries. The department managers, however, need to know the minimum, maximum, and average salaries, grouped by department. Use the Oracle database to prepare a view that displays the needed information for department managers.

```
CREATE VIEW view_dept_salary_summary AS
SELECT department_id AS "Department",
       MIN(salary) AS "Min Salary",
       MAX(salary) AS "Max Salary",
       AVG(salary) AS "Avg Salary"
FROM employees
GROUP BY department_id;
```

EXERCISE-14

OTHER DATABASE OBJECTS

Objectives

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

Database Objects

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

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

The CREATE SEQUENCE Statement Syntax

Define a sequence to generate sequential numbers automatically:

CREATE SEQUENCE *sequence*
[INCREMENT BY *n*]
[START WITH *n*]
[{MAXVALUE *n* | NOMAXVALUE}]
[{MINVALUE *n* | NOMINVALUE}]
[{CYCLE | NOCYCLE}] [{CACHE
n | NOCACHE}];
In the syntax:
sequence is the name of the sequence generator

INCREMENT BY *n* specifies the interval between sequence numbers where *n* is an integer (If this clause is omitted, the sequence increments by 1.)

START WITH *n* specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)

MAXVALUE *n* specifies the maximum value the sequence can generate
NOMAXVALUE specifies a maximum value of 10^27 for an ascending sequence and –1 for a descending sequence (This is the default option.)

MINVALUE *n* specifies the minimum sequence value
NOMINVALUE specifies a minimum value of 1 for an ascending sequence and – (10^26) for a descending sequence (This is the default option.)

CYCLE | NOCYCLE specifies whether the sequence continues to generate values after reaching its maximum or minimum value (NOCYCLE is the default option.)

CACHE *n* | NOCACHE specifies how many values the Oracle server preallocates and keep in memory (By default, the Oracle server caches 20 values.)

Creating a Sequence

• Create a sequence named DEPT_DEPTID_SEQ to be used for the primary key of the DEPARTMENTS table.
• Do not use the CYCLE option.

EXAMPLE:

```
CREATE SEQUENCE dept_deptid_seq
INCREMENT BY 10
START WITH 120
MAXVALUE 9999
NOCACHE
NOCYCLE;
```

Confirming Sequences

• Verify your sequence values in the USER_SEQUENCES data dictionary table.

• The LAST_NUMBER column displays the next available sequence number if NOCACHE is specified.

EXAMPLE:

```
SELECT sequence_name, min_value, max_value, increment_by, last_number
```

NEXTVAL and CURRVAL Pseudocolumns

• NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.

• CURRVAL obtains the current sequence value.

• NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Rules for Using NEXTVAL and CURRVAL

You can use NEXTVAL and CURRVAL in the following contexts:

• The SELECT list of a SELECT statement that is not part of a subquery
• The SELECT list of a subquery in an INSERT statement
• The VALUES clause of an INSERT statement
• The SET clause of an UPDATE statement

You cannot use NEXTVAL and CURRVAL in the following contexts:

• The SELECT list of a view
• A SELECT statement with the DISTINCT keyword
• A SELECT statement with GROUP BY, HAVING, or ORDER BY clauses
• A subquery in a SELECT, DELETE, or UPDATE statement
• The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

Using a Sequence

• Insert a new department named “Support” in location ID 2500.

• View the current value for the DEPT_DEPTID_SEQ sequence.

EXAMPLE:

```
INSERT INTO departments(department_id, department_name, location_id)
VALUES (dept_deptid_seq.NEXTVAL, 'Support', 2500);

SELECT dept_deptid_seq.CURRVAL FROM dual;
```

The example inserts a new department in the DEPARTMENTS table. It uses the DEPT_DEPTID_SEQ sequence for generating a new department number as follows:
You can view the current value of the sequence:
SELECT dept_deptid_seq.CURRVAL FROM dual; **Removing a Sequence**
• Remove a sequence from the data dictionary by using the DROP SEQUENCE statement.

• Once removed, the sequence can no longer be referenced.
EXAMPLE:

```
DROP SEQUENCE dept_deptid_seq;
```

What is an Index?

An index:
• Is a schema object
• Is used by the Oracle server to speed up the retrieval of rows by using a pointer
• Can reduce disk I/O by using a rapid path access method to locate data quickly
• Is independent of the table it indexes
• Is used and maintained automatically by the Oracle server

How Are Indexes Created?

• Automatically: A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
• Manually: Users can create nonunique indexes on columns to speed up access to the rows.

Types of Indexes

Two types of indexes can be created. One type is a unique index: the Oracle server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE key constraint. The name of the index is the name given to the constraint.

The other type of index is a nonunique index, which a user can create. For example, you can create a FOREIGN KEY column index for a join in a query to improve retrieval speed.

Creating an Index

• Create an index on one or more columns.

• Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table.

CREATE INDEX *index*
ON *table* (*column*[, *column*]...);
EXAMPLE:
CREATE INDEX emp_last_name_idx
ON employees(last_name); **In the**
syntax: *index* is the name of the index
table is the name of the table
column is the name of the column in the table to be indexed

When to Create an Index

You should create an index if:
• A column contains a wide range of values
• A column contains a large number of null values
• One or more columns are frequently used together in a WHERE clause or a join condition
• The table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows

When Not to Create an Index
It is usually not worth creating an index if:

• The table is small
• The columns are not often used as a condition in the query
• Most queries are expected to retrieve more than 2 to 4 percent of the rows in the table • The table is updated frequently
• The indexed columns are referenced as part of an Expression **Confirming Indexes**
• The USER_INDEXES data dictionary view contains the name of the index and its uniqueness.
• The USER_IND_COLUMNS view contains the index name, the table name, and the column name.

EXAMPLE:

```
SELECT ic.index_name, ic.column_name, ic.column_position col_pos,ix.uniqueness
FROM user_indexes ix, user_ind_columns ic WHERE
ic.index_name = ix.index_name
AND ic.table_name = 'EMPLOYEES';
```

Removing an Index

• Remove an index from the data dictionary by using the DROP INDEX command.

• Remove the UPPER_LAST_NAME_IDX index from the data dictionary.

• To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

```
DROP INDEX upper_last_name_idx;
```

```
DROP INDEX index;
```

Find the Solution for the following:

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by ten numbers. Name the sequence DEPT_ID_SEQ.

```
CREATE SEQUENCE DEPT_ID_SEQ
START WITH 200
INCREMENT BY 10
MAXVALUE 1000
NOCYCLE
```

2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number

```
SELECT sequence_name, max_value,
increment_by, last_number
FROM user_sequences
WHERE sequence_name = 'DEPT_ID_SEQ'
```

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.

```
INSERT INTO DEPARTMENT (DEPT_ID, DEPT_NAME)
VALUES (DEPT_ID_SEQ.NEXTVAL, 'Education');

INSERT INTO DEPARTMENT (DEPT_ID, DEPT_NAME)
VALUES (DEPT_ID_SEQ.NEXTVAL, 'Administration');
```

4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.

```
SELECT * FROM DEPARTMENT
WHERE DEPT_NAME
IN ('Education', 'Administration');
```

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

```
SELECT ic.index_name, ic.column_name, ic.column_position
AS col_pos, ix.uniqueness
FROM user_indexes ix
JOIN user_ind_columns ic
ON ic.index_name = ix.index_name
WHERE ic.table_name = 'EMP';
```

Download Execution time: 0.168 seconds

INDEX_NAME	COLUMN_NAME	COL_POS	UNIQUENESS
No items to display.			

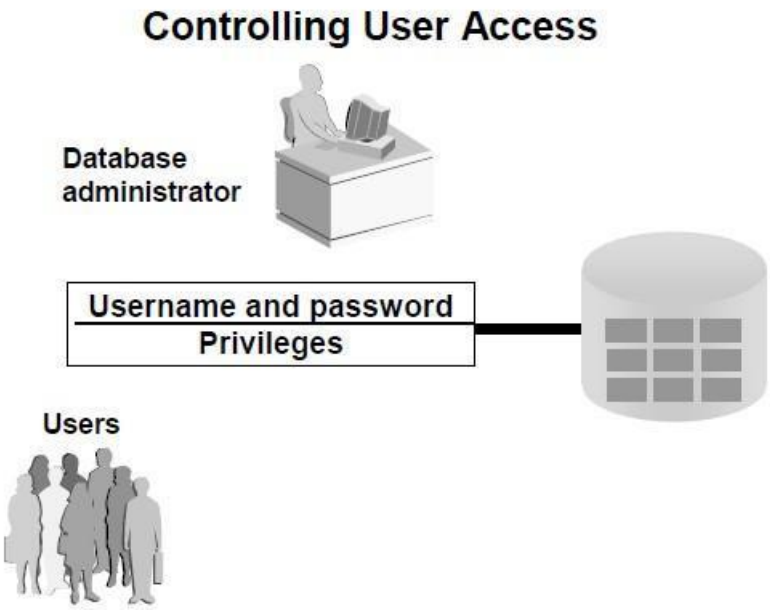
EXERCISE-15

Controlling User Access

Objectives

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

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



Controlling User Access

In a multiple-user environment, you want to maintain security of the database access and use. With Oracle server database security, you can do the following:
• Control database access
• Give access to specific objects in the database
• Confirm given and received *privileges* with the Oracle data dictionary
• Create synonyms for database objects

Privileges

• Database security: – System security
– Data security
• System privileges: Gaining access to the database
• Object privileges: Manipulating the content of the database objects
• Schemas: Collections of objects, such as tables, views, and sequences

System Privileges

• More than 100 privileges are available.
• The database administrator has high-level system privileges for tasks such as:
– Creating new users
– Removing users
– Removing tables
– Backing up tables

Typical DBA Privileges

System Privilege	Operations Authorized
CREATE USER	Grantee can create other Oracle users (a privilege required for a DBA role).
DROP USER	Grantee can drop another user.
DROP ANY TABLE	Grantee can drop a table in any schema.
BACKUP ANY TABLE	Grantee can back up any table in any schema with the export utility.
SELECT ANY TABLE	Grantee can query tables, views, or snapshots in any schema.
CREATE ANY TABLE	Grantee can create tables in any schema.

Creating Users

The DBA creates users by using the CREATE USER statement.

EXAMPLE:

CREATE USER scott IDENTIFIED BY tiger;

User System Privileges

• Once a user is created, the DBA can grant specific system privileges to a user.
• An application developer, for example, may have the following system privileges:
– CREATE SESSION
– CREATE TABLE
– CREATE SEQUENCE
– CREATE VIEW
– CREATE PROCEDURE

GRANT *privilege* [, *privilege*...]
TO *user* [, *user*| *role, PUBLIC*...];

Typical User Privileges

System Privilege	Operations Authorized
CREATE SESSION	Connect to the database
CREATE TABLE	Create tables in the user's schema
CREATE SEQUENCE	Create a sequence in the user's schema
CREATE VIEW	Create a view in the user's schema
CREATE PROCEDURE	Create a stored procedure, function, or package in the user's schema

.

In the syntax:

privilege is the system privilege to be granted

user |role|PUBLIC is the name of the user, the name of the role, or PUBLIC designates that every user is granted the privilege

Note: Current system privileges can be found in the dictionary view SESSION_PRIVS.

Granting System Privileges

The DBA can grant a user specific system privileges.

GRANT create session, create table, create sequence, create view TO scott;

What is a Role?

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges.

A user can have access to several roles, and several users can be assigned the same role. Roles are typically created for a database application.

Creating and Assigning a Role

First, the DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Syntax
CREATE ROLE *role*;
In the syntax: *role* is the name of the
role to be created
Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.

Creating and Granting Privileges to a Role

CREATE ROLE manager;
Role created.

GRANT create table, create view TO manager; Grant
succeeded.

GRANT manager TO DEHAAN, KOCHHAR;
Grant succeeded.

• Create a role
• Grant privileges to a role
• Grant a role to users

Changing Your Password

• The DBA creates your user account and initializes your password.

• You can change your password by using the

ALTER USER statement.

ALTER USER scott IDENTIFIED
BY lion;
User altered.

Object Privileges

Object Privilege	Table	View	Sequence	Procedure
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√	√		
SELECT	√	√	√	
UPDATE	√	√		

Object Privileges

• Object privileges vary from object to object.

• An owner has all the privileges on the object.

• An owner can give specific privileges on that owner's object. GRANT *object_priv* [(*columns*)]
ON *object*
TO {*user*|*role*|PUBLIC}
[WITH GRANT OPTION];

In the syntax:

object_priv is an object privilege to be granted

ALL specifies all object privileges

columns specifies the column from a table or view on which privileges are granted

ON *object* is the object on which the privileges are granted

TO identifies to whom the privilege is granted

PUBLIC grants object privileges to all users

WITH GRANT OPTION allows the grantee to grant the object privileges to other users and roles

Granting Object Privileges

• Grant query privileges on the EMPLOYEES table.

• Grant privileges to update specific columns to users and roles.

```
GRANT select
ON employees
TO sue, rich;

GRANT update (department_name, location_id)
ON departments
TO scott, manager;
```

Using the WITH GRANT OPTION and PUBLIC Keywords

• Give a user authority to pass along privileges.

• Allow all users on the system to query data from Alice's DEPARTMENTS table.

```
GRANT select, insert
ON departments
TO scott
WITH GRANT OPTION;
.
GRANT select
ON alice.departments
TO PUBLIC;
```

How to Revoke Object Privileges

• You use the REVOKE statement to revoke privileges granted to other users.

• Privileges granted to others through the WITH GRANT OPTION clause are also revoked.

```
REVOKE {privilege [, privilege...]|ALL}
ON object
FROM {user[, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

In the syntax:

CASCADE is required to remove any referential integrity constraints made to the CONSTRAINTS object by means of the REFERENCES privilege

Revoking Object Privileges

As user Alice, revoke the SELECT and INSERT privileges given to user Scott on the DEPARTMENTS table.

```
REVOKE select, insert
ON departments
FROM scott;
```

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?
   To log on to the Oracle Server, a user must be granted the CREATE SESSION privilege. This is a system privilege allowing a user to establish a connection to the database and start a session.

---

2. What privilege should a user be given to create tables?
   To create tables, a user must have the CREATE TABLE privilege, which is also a syster privilege enabling the user to create new tables within their schema or database.

---

3. If you create a table, who can pass along privileges to other users on your table?
   When you create a table, you as the owner have all privileges on that table, including the ability to grant (pass along) privileges to other users on your table. Only the owner or a user with special privileges can grant such object privileges.

---

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 creating many users needing the same system privileges, you should create a role. A role is a named group of privileges that can be granted to many users, simplifying privilege management.

---

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

```
ALTER USER your_username IDENTIFIED BY new_password
```

---

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 your_schema.departments
TO other_user WITH GRANT OPTION;
```

7. Query all the rows in your DEPARTMENTS table.

```
SELECT * FROM your_schema.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.

```
INSERT INTO departments (department_id, department_name)
VALUES (500, 'Education');

INSERT INTO departments (department_id, department_name)
VALUES (510, 'Human Resources');
```

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

```
SELECT * FROM other_team_schema.department;
```

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

```
REVOKE SELECT ON your_schema.departments
FROM other_user;
```

Evaluation Procedure	Marks awarded
Practice Evaluation (5)	
Viva(5)	
Total (10)	
Faculty Signature	

PL/SQL

PL/SQL

Control Structures

In addition to SQL commands,PL/SQL can also process data usin flow of statements.the flow of control statements are classified into the following categories.

- Conditional control - Branching
- Iterative control - looping
- Sequential control

BRANCHING in PL/SQL:

Sequence of statements can be executed on satisfying certain condition .

If statements are being used and different forms of if are:

1.Simple IF

2.ELSIF

3.ELSE IF ***SIMPLE***

IF:

Syntax:

```
IF condition THEN
   statement1;
statement2;

END IF;
```

IF-THEN-ELSE STATEMENT:

Syntax:

```
IF condition THEN
statement1;

ELSE
   statement2;
END IF;
```

ELSIF STATEMENTS:

Syntax:

```
IF condition1 THEN
   statement1;
ELSIF condition2 THEN
   statement2;
ELSIF condition3 THEN
   statement3;
ELSE
   statementn;
END IF;
```

NESTED IF :

Syntax:

```
IF
condition THEN
    statement1;
ELSE
  IF condition THEN
     statement2;
ELSE
     statement3;
END IF;

END IF; ELSE
  statement3;
END IF;
```

SELECTION IN PL/SQL(Sequential Controls)

SIMPLE CASE

Syntax:

```
CASE SELECTOR
    WHEN Expr1 THEN statement1;
WHEN Expr2 THEN statement2;

:
ELSE
     Statement n;
END CASE;
```

SEARCHED CASE:

```
CASE
  WHEN  searchcondition1 THEN statement1;
WHEN  searchcondition2 THEN statement2;

:

:

ELSE
statementn;
END CASE;
```

ITERATIONS IN PL/SQL

Sequence of statements can be executed any number of times using loop construct.

It is broadly classified into:

- Simple Loop
- For Loop
- While Loop

SIMPLE LOOP

Syntax: LOOP

```
 statement1;
```

EXIT [WHEN Condition];

END LOOP; ***WHILE***

LOOP

Syntax:

WHILE condition LOOP

statement1; statement2;

END LOOP;

FOR LOOP

Syntax:

FOR counter IN [REVERSE]

LowerBound..UpperBound LOOP

statement1; statement2;

END LOOP;

PROGRAM 1

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

```
DECLARE
V_SAL EMPP.SALARY%TYPE;
V_COMM EMPP.COMMISSION_PCT%TYPE;
V_INCENTIVE NUMBER;
BEGIN
SELECT SALARY,COMMISSION_PCT INTO V_SAL,V_COMM FROM EMPP WHERE EMPLOYEE_ID=110;
V_INCENTIVE:=V_SAL*NVL(V_COMM,0);
DBMS_OUTPUT.PUT_LINE('INCENTIVE:'|| V_INCENTIVE);
END;
```

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

```
DECLARE
MYVAR NUMBER:=100;
"MYVAR" NUMBER:=200;
BEGIN
DBMS_OUTPUT.PUT_LINE('UNQUOTED MYVAR:'||MYVAR);
DBMS_OUTPUT.PUT_LINE('QUOTED MYVAR:'||MYVAR);
DBMS_OUTPUT.PUT_LINE('QUOTED "MYVAR":'||"MYVAR");
END;
```

PROGRAM 3

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

Sample table: employees

```
DECLARE
V_OLDSAL EMPP.SALARY%TYPE;
V_NEWSAL EMPP.SALARY%TYPE;
BEGIN
SELECT SALARY INTO V_OLDSAL FROM EM
V_NEWSAL :=V_OLDSAL*1.10;
UPDATE EMPP SET SALARY=V_NEWSAL WHE
DBMS_OUTPUT.PUT_LINE('EMPLOYEE_ID:'
DBMS_OUTPUT.PUT_LINE('OLD SALARY:'|
DBMS_OUTPUT.PUT_LINE('NEW SALARY:'|
END;
```

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.

```
CREATE OR REPLACE PROCEDURE CH_EMP_DET(EMP_ID IN EMPP.EMPLOYEE_ID%TYPE) IS
V_SAL EMPP.SALARY%TYPE;
V_JOB_ID EMPP.JOB_ID%TYPE;
BEGIN
SELECT SALARY,JOB_ID INTO V_SAL,V_JOB_ID FROM EMPP WHERE EMPLOYEE_ID=EMP_ID;
IF(V_SAL IS NOT NULL)AND(V_JOB_ID IS NOT NULL) THEN DBMS_OUTPUT.PUT_LINE('EMP HAS BOTH SAL AND JOBID');
ELSIF (V_SAL IS NOT NULL)AND(V_JOB_ID IS NULL) THEN DBMS_OUTPUT.PUT_LINE('EMP HAS SAL BUT NO JOBID');
ELSIF(V_SAL IS NULL)AND(V_JOB_ID IS NOT NULL) THEN DBMS_OUTPUT.PUT_LINE('EMP HAS JOBID BUT NO SAL');
ELSE DBMS_OUTPUT.PUT_LINE('EMP HAS NEITHER SAL NOR JOB_ID');
END IF;
END;
```

PROGRAM 5

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

```
DECLARE
V_NAME VARCHAR2(25);
BEGIN
V_NAME:='KING_100%';
IF V_NAME LIKE 'KING%' THEN DBMS_OUTPUT.PUT_LINE('NAME STARTS WITH KING')
END IF;
IF V_NAME LIKE 'KIN__100%' THEN DBMS_OUTPUT.PUT_LINE('NAME MATCHES KIN__100%');
END IF;
IF V_NAME LIKE 'KING\_%' ESCAPE '\' THEN DBMS_OUTPUT.PUT_LINE('NAME MATCHES KING\_%');
END IF;
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.

```
DECLARE
NUM1 NUMBER:=45;
NUM2 NUMBER:=20;
NUM_SMALL NUMBER;
NUM_LARGE NUMBER;
BEGIN
IF NUM1<NUM2 THEN NUM_SMALL:=NUM1;
NUM_SMALL:=NUM1;
NUM_LARGE:=NUM2;
ELSE
NUM_SMALL:=NUM2;
NUM_LARGE:=NUM1;
END IF;
```

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.

```
CREATE OR REPLACE PROCEDURE CLACINC(EMP_ID IN EMPP.EMPLOYEE_ID%TYPE,SALES_ACH IN NUMBER) IS
P_INC NUMBER;
BEGIN
IF SALES_ACH<50000 THEN
P_INC:=0;
ELSIF SALES_ACH BETWEEN 50000 AND 100000 THEN
P_INC:=SALES_ACH*0.05;
ELSE
P_INC:=SALES_ACH*0.15;
END IF;
DBMS_OUTPUT.PUT_LINE('INCENTIVE:'|| P_INC);
END;
```

PROGRAM 8

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

```
CREATE OR REPLACE PROCEDURE CLACINC(EMP_ID IN EMPP.EMPLOYEE_ID%TYPE,SALES_ACH IN NUMBER) IS
P_INC NUMBER;
BEGIN
IF SALES_ACH<50000 THEN
P_INC:=0;
ELSIF SALES_ACH BETWEEN 50000 AND 100000 THEN
P_INC:=SALES_ACH*0.05;
ELSE
P_INC:=SALES_ACH*0.15;
END IF;
DBMS_OUTPUT.PUT_LINE('INCENTIVE:'|| P_INC);
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.

```
DECLARE
EMP_C NUMBER;
C_MAX NUMBER:=45;
V_VAN NUMBER;
BEGIN
SELECT COUNT(*) INTO EMP_C FROM EMPP WHERE DEPARTMENT_ID=80;
V_VAN:=C_MAX-EMP_C;
IF V_VAN>0 THEN
DBMS_OUTPUT.PUT_LINE('DEPATMENT 80 HAS '||V_VAN||' VACANCIES');
ELSE
DBMS_OUTPUT.PUT_LINE('DEPATMENT 80 HAS NO VACANCIES');
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.

```
CREATE OR REPLACE PROCEDURE CK_VAC(DEPT_ID IN EMPP.DEPARTMENT_ID%TYPE,MAX_COUNT IN NUMBER) IS
V_COUNT NUMBER;
V_VACANCIES NUMBER;
BEGIN
SELECT COUNT(*) INTO V_COUNT FROM EMPP WHERE DEPARTMENT_ID=DEPT_ID;
V_VACANCIES:=MAX_COUNT - V_COUNT;
IF(V_VACANCIES >0) THEN DBMS_OUTPUT.PUT_LINE('VACANCIES AVAILABLE IN'||DEPT_ID||'IS'||V_VACANCIES);
ELSE  DBMS_OUTPUT.PUT_LINE('NO VACANCIES AVAILABLE');
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.

```
BEGIN
FOR REC IN(SELECT EMPLOYEE_ID,LAST_NAME,JOB_ID,HIRE_DATE,SALARY FROM EMPP)
LOOP
DBMS_OUTPUT.PUT_LINE('ID:'||REC.EMPLOYEE_ID||' NAME: '||REC.LAST_NAME||' JOB: '||REC.JOB_ID||' HIRE DATE: '||REC.HIRE_DATE||' SALARY: '||REC.SALARY);
END LOOP;
END;
```

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

```
BEGIN
FOR REC IN(SELECT EMPLOYEE_ID,LAST_NAME,D.DEPT_NAME FROM EMPP E JOIN DEPTT D ON E.DEPARTMENT_ID=D.DEPT_ID)
LOOP
DBMS_OUTPUT.PUT_LINE('ID: '||REC.EMPLOYEE_ID||' NAME: '||REC.LAST_NAME||' DEPARTMENT NAME: '||REC.DEPT_NAME);
END LOOP;
END;
```

PROGRAM 17

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

```
FOR REC IN(SELECT E.EMPLOYEE_ID,E.LAST_NAME,J.START_DATE FROM EMPP E JOIN JOB_HISTORY J ON E.EMPLOYEE_ID=J.EMPLOYEE_ID)
LOOP
DBMS_OUTPUT.PUT_LINE('ID:'||REC.EMPLOYEE_ID||' NAME: '||REC.LAST_NAME||' HIS/HER JOB START DATE:'||REC.START_DATE);
END LOOP;
END;
```

PROGRAM 18

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

```
BEGIN
FOR REC IN(SELECT E.EMPLOYEE_ID,E.LAST_NAME,J.START_DATE FROM EMPP E JOIN JOB_HISTORY J ON E.EMPLOYEE_ID=J.EMPLOYEE_ID)
LOOP
DBMS_OUTPUT.PUT_LINE('ID:'||REC.EMPLOYEE_ID||' NAME: '||REC.LAST_NAME||' HIS/HER 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.

```
BEGIN
FOR REC IN(SELECT E.EMPLOYEE_ID,E.LAST_NAME,J.END_DATE FROM EMPP E JOIN JOB_HISTORY J ON E.EMPLOYEE_ID=J.EMPLOYEE_ID)
LOOP
DBMS_OUTPUT.PUT_LINE('ID:'||REC.EMPLOYEE_ID||' NAME: '||REC.LAST_NAME||' HIS/HER JOB END DATE:'||REC.END_DATE);
END LOOP;
END;
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	

PROGRAM 20

Total (15)	
Faculty Signature	

EXERCISE-16

PROCEDURES AND FUNCTIONS

PROCEDURES

DEFINITION

A procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task. They are essentially sub-programs. Procedures and functions are made up of,

- Declarative part
- Executable part
- Optional exception handling part

These procedures and functions do not show the errors.

KEYWORDS AND THEIR PURPOSES

REPLACE: It recreates the procedure if it already exists.

PROCEDURE: It is the name of the procedure to be created.

ARGUMENT: It is the name of the argument to the procedure. Paranthesis can be omitted if no arguments are present.

IN: Specifies that a value for the argument must be specified when calling the procedure ie . used to pass values to a sub-program. This is the default parameter.

OUT: Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie. used to return values to a caller of the sub-program.

INOUT: Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it's calling environment after execution.

RETURN: It is the datatype of the function's return value because eve ry function must return a value, this clause is required.

PROCEDURES – SYNTAX

```
create or replace procedure <procedure name> (argument {in,out,inout} datatype ) {is,as}
variable declaration;   constant declaration;   begin
PL/SQL subprogram body;
exception
exception PL/SQL block;
end;
```

FUNCTIONS – SYNTAX

```
create or replace function <function name> (argument in datatype,……) return datatype {is,as}
variable declaration;
 constant declaration;
begin
 PL/SQL subprogram body;
exception
 exception PL/SQL block;
 end;
```

CREATING THE TABLE 'ITITEMS' AND DISPLAYING THE CONTENTS

SQL> create table ititems(itemid number(3), actualprice number(5), ordid number(4), prodid number(4)); Table created.

SQL> insert into ititems values(101, 2000, 500, 201); 1 row created.

SQL> insert into ititems values(102, 3000, 1600, 202); 1 row created.

SQL> insert into ititems values(103, 4000, 600, 202); 1 row created.

SQL> select * from ititems;

ITEMID	ACTUALPRICE	ORDID	PRODID
101	2000	500	201
102	3000	1600	202
103	4000	600	202

PROGRAM FOR GENERAL PROCEDURE – SELECTED RECORD'S PRICE IS INCREMENTED BY 500 , EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE

```
SQL> create procedure itsum(identity number, total number) is price number;
2  null_price exception;
3  begin
4  select actualprice into price from ititems where itemid=identity;
5  if price is null then
6  raise null_price;
7  else
8  update ititems set actualprice=actualprice+total where itemid=identity;   9  end if;
10  exception
11  when null_price then
12  dbms_output.put_line('price is null');
13  end;  14  /
Procedure created.
```

SQL> exec itsum(101, 500);
PL/SQL procedure successfully completed.

```
SQL> select * from ititems;
ITEMID   ACTUALPRICE      ORDID     PRODID
---------          -----------          ---------     ---------
         101           2500                 500         201
         102           3000                1600         202
         103           4000                 600          202
```

PROCEDURE FOR 'IN' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;

SQL> create procedure yyy (a IN number) is price number;
2  begin
3  select actualprice into price from ititems where itemid=a;
4  dbms_output.put_line('Actual price is ' || price);
5  if price is null then
6  dbms_output.put_line('price is null');
7  end if;
8  end;   9  /
Procedure created.

SQL> exec yyy(103);
Actual price is 4000
PL/SQL procedure successfully completed.
```

PROCEDURE FOR 'OUT' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;

SQL> create procedure zzz (a in number, b out number) is identity number;
2  begin
3   select ordid into identity from ititems where itemid=a;
4   if identity<1000 then
5   b:=100;
6   end if;
7   end;   8  /
Procedure created.

SQL> declare
2  a number;
3  b number;
4  begin
5  zzz(101,b);
6  dbms_output.put_line('The value of b is '|| b);
7  end;
8  /
The value of b is 100
PL/SQL procedure successfully completed.
```

PROCEDURE FOR 'INOUT' PARAMETER – CREATION, EXECUTION

```
SQL> create procedure itit ( a in out number) is
2  begin
3  a:=a+1;
4  end;   5  /
  Procedure created.

SQL> declare
2  a number:=7;
3  begin
4  itit(a);
5  dbms_output.put_line('The updated value is '||a);
6  end;
7  /
The updated value is 8
PL/SQL procedure successfully completed.
```

CREATE THE TABLE 'ITTRAIN' TO BE USED FOR FUNCTIONS

```
SQL>create table ittrain ( tno number(10), tfare number(10)); Table
created.

SQL>insert into ittrain values (1001, 550); 1 row
created.

SQL>insert into ittrain values (1002, 600); 1 row
created.

SQL>select * from ittrain;
    TNO       TFARE
  ---------    ------------
1001         550
1002         600
```

PROGRAM FOR FUNCTION AND IT'S EXECUTION

```
SQL> create function aaa (trainnumber number) return number is
2  trainfunction ittrain.tfare % type;
3  begin
4  select tfare into trainfunction from ittrain where tno=trainnumber;
5  return(trainfunction);
6  end;
7  /
Function created.

SQL> set serveroutput on;

SQL> declare
2  total number;
3  begin
```

```
4  total:=aaa (1001);
5  dbms_output.put_line('Train fare is Rs. '||total);
6  end;
7  /
Train fare is Rs.550
PL/SQL procedure successfully completed.
```

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

```
CREATE OR REPLACE FUNCTION FACTORIAL(N IN NUMBER) RETURN NUMBER IS
RESULT NUMBER:=1;
BEGIN
FOR I IN 1..N
LOOP
RESULT :=RESULT*I;
END LOOP;
RETURN 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 (B_ID IN NUMBER,B_NAME OUT VARCHAR2,B_PRICE IN OUT NUMBER) IS
BEGIN
SELECT BOOK_NAME,PRICE INTO B_NAME,B_PRICE FROM LIBRARY WHERE BOOK_ID=B_ID;
EXCEPTION
WHEN NO_DATA_FOUND THEN
B_NAME:='NO SUCH BOOK FOUND';
B_PRICE:=0;
END;
/
```

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

EXERCISE-17

TRIGGER

DEFINITION

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

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

The different uses of triggers are as follows,

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

TYPES OF TRIGGERS

The various types of triggers are as follows,

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

VARIABLES USED IN TRIGGERS

- :new
- :old

These two variables retain the new and old values of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation

SYNTAX

```
 create or replace trigger triggername [before/after] {DML statements}
on [tablename] [for each row/statement]  begin
 ------------------------
```

```
 ------------------------
 ------------------------
exception end;
```

USER DEFINED ERROR MESSAGE

The package “raise_application_error” is used to issue the user defined error messages

Syntax: raise_application_error(error number,‘error message‘);

The error number can lie between -20000 and -20999.

The error message should be a character string.

TO CREATE THE TABLE ‘ITEMPLS’

SQL> create table itempls (ename varchar2(10), eid number(5), salary number(10)); Table created.

SQL> insert into itempls values('xxx',11,10000); 1 row created.

SQL> insert into itempls values('yyy',12,10500); 1 row created.

SQL> insert into itempls values('zzz',13,15500); 1 row created.

```
SQL> select * from itempls;
ENAME          EID    SALARY
---------- --------- --------- xxx
11     10000 yyy              12
10500
zzz             13     15500
```

TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE

```
SQL> create trigger ittrigg before insert or update or delete on itempls for each row
2  begin
3  raise_application_error(-20010,'You cannot do manipulation');
4  end;
  5
  6  /
Trigger created.

SQL> insert into itempls values('aaa',14,34000);
insert into itempls values('aaa',14,34000)
*
```

```
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

SQL> delete from itempls where ename='xxx';
delete from itempls where ename='xxx'
*
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'

SQL> update itempls set eid=15 where ename='yyy'; update
itempls set eid=15 where ename='yyy'
*
ERROR at line 1:
ORA-20010: You cannot do manipulation
ORA-06512: at "STUDENT.ITTRIGG", line 2
ORA-04088: error during execution of trigger 'STUDENT.ITTRIGG'
```

TO DROP THE CREATED TRIGGER

```
SQL> drop trigger ittrigg;

Trigger dropped.
```

TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE AND DOES NOT ALLOW UPDATION AND INSERTION

```
SQL> create trigger ittriggs before insert or update of salary on itempls for each row
2  declare
3  triggsal itempls.salary%type;
4  begin
5  select salary into triggsal from itempls where eid=12;
6  if(:new.salary>triggsal or :new.salary<triggsal) then
7  raise_application_error(-20100,'Salary has not been changed');   8  end if;
9    end;
10  /

Trigger created.

SQL> insert into itempls values ('bbb',16,45000);
insert into itempls values ('bbb',16,45000)
            *
ERROR at line 1:
ORA-04098: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
```

```
SQL> update itempls set eid=18 where ename='zzz'; update
itempls set eid=18 where ename='zzz'
          *
ERROR at line 1:
ORA-04298: trigger 'STUDENT.ITTRIGGS' is invalid and failed re-validation
```

Cursor for loop

- Explicit cursor
- Implicit cursor

TO CREATE THE TABLE 'SSEMPP'

```
SQL> create table ssempp( eid number(10), ename varchar2(20), job varchar2(20), sal number
(10),dnonumber(5)); Table
created.

SQL> insert into ssempp values(1,'nala','lecturer',34000,11); 1 row
created.

SQL> insert into ssempp values(2,'kala',' seniorlecturer',20000,12); 1 row
created.

SQL> insert into ssempp values(5,'ajay','lecturer',30000,11); 1 row
created.

SQL> insert into ssempp values(6,'vijay','lecturer',18000,11); 1 row
created.

SQL> insert into ssempp values(3,'nila','professor',60000,12); 1 row
created.

SQL> select * from ssempp;
   EID    ENAME              JOB                  SAL       DNO
--------- -------------------- --------------------     ---------     ---------
1       nala                 lecturer                34000        11
       2       kala                 seniorlecturer          20000         12
5       ajay                 lecturer                30000         11
6       vijay                lecturer                 18000         11
       3       nila                 professor               60000         12
```

EXTRA PROGRAMS

TO WRITE A PL/SQL BLOCK TO DISPLAY THE EMPOYEE ID AND EMPLOYEE NAME USING CURSOR FOR LOOP

```
SQL> set serveroutput on;
SQL> declare
2  begin
```

```
3  for emy in (select eid,ename from ssempp)
4  loop
5  dbms_output.put_line('Employee id and employee name  are '|| emy.eid ‘and’|| emy.ename);   6
   end loop;
7  end;
8  /
Employee id and employee name are 1 and nala
Employee id and employee name are 2 and kala
Employee id and employee name are 5 and ajay
Employee id and employee name are 6 and vijay
Employee id and employee name are 3 and nila

PL/SQL procedure successfully completed.
```

TO WRITE A PL/SQL BLOCK TO UPDATE THE SALARY OF ALL EMPLOYEES WHERE DEPARTMENT NO IS 11 BY 5000 USING CURSOR FOR LOOP AND TO DISPLAY THE UPDATED TABLE

```
SQL> set serveroutput on;
SQL> declare
  2  cursor cem is select eid,ename,sal,dno from ssempp where dno=11;
3  begin
4    --open cem;
5    for rem in cem
6    loop
7    update ssempp set sal=rem.sal+5000 where eid=rem.eid;
8    end loop;
9    --close cem;
10  end;
11  /
```

PL/SQL procedure successfully completed.

```
SQL> select * from ssempp;
```

EID	ENAME	JOB	SAL	DNO
1	nala	lecturer	39000	11
2	kala	seniorlecturer	20000	12
5	ajay	lecturer	35000	11
6	vijay	lecturer	23000	11
3	nila	professor	60000	12

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

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

SQL> /
Employee code and employee salary are 1 and 39000
```

Employee code and employee salary are 5 and 35000
Employee code and employee salary are 6 and 23000

PL/SQL procedure successfully completed.

TO WRITE A PL/SQL BLOCK TO UPDATE THE SALARY BY 5000 WHERE THE JOB IS LECTURER , TO CHECK IF UPDATES ARE MADE USING IMPLICIT CURSORS AND TO DISPLAY THE UPDATED TABLE

```
SQL> declare
2  county number;
3  begin
4  update ssempp set sal=sal+10000 where job='lecturer';
5  county:= sql%rowcount;
6  if county > 0 then
7  dbms_output.put_line('The number of rows are '|| county);   8  end if;
9    if sql %found then
10  dbms_output.put_line('Employee record modification successful');
11  else if sql%notfound then
12  dbms_output.put_line('Employee record is not found');
13  end if;
14  end if;
15  end;
16  /
```

The number of rows are 3

Employee record modification successful

PL/SQL procedure successfully completed.

SQL> select * from ssempp;

EID	ENAME	JOB	SAL	DNO
1	nala	lecturer	44000	11
2	kala	seniorlecturer	20000	12
5	ajay	lecturer	40000	11
6	vijay	lecturer	28000	11
3	nila	professor	60000	12

PROGRAMS

TO DISPLAY HELLO MESSAGE

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

```
6  end;
7  /
Hello

PL/SQL procedure successfully completed.
```

TO INPUT A VALUE FROM THE USER AND DISPLAY IT

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

PL/SQL procedure successfully completed.
```

GREATEST OF TWO NUMBERS

```
SQL> set serveroutput on;

SQL> declare
  2  a number(7);
3  b number(7);
4    begin
5    a:=&a;
6    b:=&b;
7    if(a>b) then
8    dbms_output.put_line (' The grerater of the two is'|| a);
9    else
10  dbms_output.put_line (' The grerater of the two is'|| b);
11  end if;
12  end;
13  /
Enter value for a: 5 old   5:
a:=&a; new   5: a:=5;
Enter value for b: 9 old   6:
b:=&b; new   6: b:=9; The
grerater of the two is9

PL/SQL procedure successfully completed.
```

GREATEST OF THREE NUMBERS

```
SQL> set serveroutput on;

SQL> declare
  2  a number(7);
3  b number(7);
4  c number(7);
5    begin
6    a:=&a;
7    b:=&b;
8    c:=&c;
```

```
9   if(a>b and a>c) then
10  dbms_output.put_line (' The greatest of the three is ' || a);
11  else if (b>c) then
12  dbms_output.put_line (' The greatest of the three is ' || b);
13  else
14  dbms_output.put_line (' The greatest of the three is ' || c);
15  end if;
16  end if;
17  end;
18  /
Enter value for a: 5
old   6: a:=&a; new
6: a:=5; Enter value
for b: 7 old      7:
b:=&b; new        7:
b:=7; Enter value
for c: 1 old      8:
c:=&c; new        8:
c:=1;
The greatest of the three is 7

PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 5 USING SIMPLE LOOP

```
SQL> set serveroutput on;

SQL> declare
2   a number:=1;
3   begin
4   loop
5   dbms_output.put_line (a);
6   a:=a+1;
7   exit when a>5;
8   end loop;
9   end;
10  /
1
2
3
4
5

PL/SQL procedure successfully completed.
```

PRINT NUMBERS FROM 1 TO 4 USING WHILE LOOP

```
SQL> set serveroutput on;

SQL> declare
2    a number:=1;
3    begin
4    while(a<5)
5    loop
6    dbms_output.put_line (a);
7    a:=a+1;
8    end loop;
9    end;
10  /
1
2
3
4
PL/SQL procedure successfully completed.
```

SQL> set serveroutput on;

PRINT NUMBERS FROM 1 TO 5 USING FOR LOOP

```
SQL> declare
2  a number:=1;
3  begin
4  for a in 1..5
5  loop
6  dbms_output.put_line (a);
7  end loop;
8  end;
9  /
1
2
3
4
5
PL/SQL procedure successfully completed.
```

SQL> set serveroutput on;

PRINT NUMBERS FROM 1 TO 5 IN REVERSE ORDER USING FOR LOOP

```
SQL> declare
2  a number:=1;
3  begin
4  for a in reverse 1..5
5  loop
6  dbms_output.put_line (a);
7  end loop;
8  end;
```

```
9  /
5
4
3
2
1
PL/SQL procedure successfully completed.
```

TO CALCULATE AREA OF CIRCLE

```
SQL> set serveroutput on;
SQL> declare
2    pi constant number(4,2):=3.14;
3    a number(20);
4    r number(20);
5    begin
6    r:=&r;
7    a:= pi* power(r,2);
8    dbms_output.put_line (' The area of circle is ' || a);
9    end;
10  /
Enter value for r: 2
old    6: r:=&r; new
6: r:=2;
The area of circle is 13
PL/SQL procedure successfully completed.
```

TO CREATE SACCOUNT TABLE

```
SQL> create table saccount  ( accno number(5), name varchar2(20), bal number(10)); Table
created.
SQL> insert into saccount values ( 1,'mala',20000); 1 row
created.
SQL> insert into saccount values (2,'kala',30000); 1 row
created.
SQL> select * from saccount;

   ACCNO NAME                 BAL
--------- -------------------- ---------
1 mala                  20000
2 kala                  30000
SQL> set serveroutput on;
SQL> declare
2    a_bal number(7);
3    a_no varchar2(20);
4    debit number(7):=2000;
5    minamt number(7):=500;
6    begin
7    a_no:=&a_no;
8    select bal into a_bal from saccount where accno= a_no;
9    a_bal:= a_bal-debit;
10  if (a_bal > minamt) then
11  update  saccount  set bal=bal-debit where accno=a_no;
```

```
12  end if;
13  end;
 14
 15  /
Enter value for a_no: 1
old     7:  a_no:=&a_no;
new   7: a_no:=1;

PL/SQL procedure successfully completed.
SQL> select * from saccount;
  ACCNO NAME                 BAL
--------- -------------------- ---------
1 mala                  18000
2 kala                  30000
```

TO CREATE TABLE SROUTES

```
SQL> create table sroutes ( rno number(5), origin varchar2(20), destination varchar2(20),
fare numbe
r(10), distance number(10)); Table
created.
```

```
SQL> insert into sroutes values ( 2, 'chennai', 'dindugal', 400,230); 1 row
created.
SQL> insert into sroutes values ( 3, 'chennai', 'madurai', 250,300); 1 row
created.
SQL> insert into sroutes values ( 6, 'thanjavur', 'palani', 350,370); 1 row
created.
SQL> select * from sroutes;

    RNO ORIGIN             DESTINATION             FARE  DISTANCE
--------- -------------------- -------------------- --------- ---------
       2 chennai              dindugal                400       230         3
chennai              madurai                 250       300
       6 thanjavur            palani                  350       370
SQL> set serveroutput on;

SQL> declare
2    route sroutes.rno % type;
3    fares sroutes.fare % type;
4    dist sroutes.distance % type;
5    begin
6    route:=&route;
7    select fare, distance into fares , dist from sroutes where rno=route;
8    if (dist < 250) then
9    update sroutes set fare=300 where rno=route;
10  else if dist between 250 and 370 then
11  update sroutes set fare=400 where rno=route;
12  else if (dist > 400) then
13  dbms_output.put_line('Sorry');
14  end if;
15  end if;
16  end if;
17  end;
18  /
Enter value for route: 3 old
6: route:=&route;
new   6: route:=3;

PL/SQL procedure successfully completed.

SQL> select * from sroutes;

    RNO ORIGIN             DESTINATION             FARE  DISTANCE
--------- -------------------- -------------------- --------- ---------
       2 chennai              dindugal                400       230         3
chennai              madurai                 400       300
       6 thanjavur            palani                  350       370
```

TO CREATE SCA LCULATE TABLE

```
SQL> create table scalculate ( radius number(3), area number(5,2)); Table
created.
SQL> desc scalculate;
 Name                                      Null?    Type  ----------------------------------------------------
-- -------- -----------------------------------
 RADIUS                                             NUMBER(3)
 AREA                                              NUMBER(5,2)

SQL> set serveroutput on;

SQL> declare
2    pi constant number(4,2):=3.14;
3    area number(5,2);
4    radius number(3);
5    begin
6    radius:=3;
7    while (radius <=7)
8    loop
9    area:= pi* power(radius,2);
10  insert into scalculate values (radius,area);
11  radius:=radius+1;
12  end loop;
13  end;
14  /

PL/SQL procedure successfully completed.

SQL> select * from scalculate;
  RADIUS      AREA
--------- ---------
         3     28.26
         4     50.24
         5     78.5
         6     113.04
         7     153.86


SQL> set serveroutput on;
```

TO CALCULATE FACTORIAL OF A GIVEN NUMBER

```
SQL> declare
2    f number(4):=1;
3    i number(4);
4    begin
5    i:=&i;
6    while(i>=1)
7    loop
```

```
8    f:=f*i;
9    i:=i-1;
10  end loop;
11  dbms_output.put_line('The value is ' || f);
12  end;
13  /
Enter value for i: 5
old   5: i:=&i; new
5: i:=5; The value
is 120

PL/SQL procedure successfully completed.
```

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 PR_REC_DEL BEFORE DELETE ON DEPTT
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM EMPP WHERE DEPARTMENT_ID = :OLD.DEPT_ID;
IF v_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'CANNOT DELETE DEPARTMENT - EMPLOYEE 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 CK_DUP BEFORE INSERT OR UPDATE ON DEPTT
FOR EACH ROW
DECLARE
V_COUNT NUMBER;
BEGIN
SELECT COUNT(*) INTO V_COUNT FROM DEPTT WHERE DEPT_NAME=:NEW.DEPT_NAME AND DEPT_ID!=:NEW.DEPT_ID;
IF V_COUNT >0 THEN
RAISE_APPLICATION_ERROR(-20003,'DUPLICATE DEPARTMENT');
END IF;
END;
/
```

Program 3

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

```
CREATE OR REPLACE TRIGGER CK_TOT BEFORE INSERT ON LIBRARY
FOR EACH ROW
DECLARE
V_TOT NUMBER;
V_LIMIT NUMBER:=5000;
BEGIN
SELECT NVL(SUM(PRICE),0)+:NEW.PRICE INTO V_TOT FROM LIBRARY;
IF V_TOT>V_LIMIT THEN
RAISE_APPLICATION_ERROR(-20004,'CANNOT INSERT TOTAL PRICE EXCEEDS');
END IF;
END;
```

Program 4

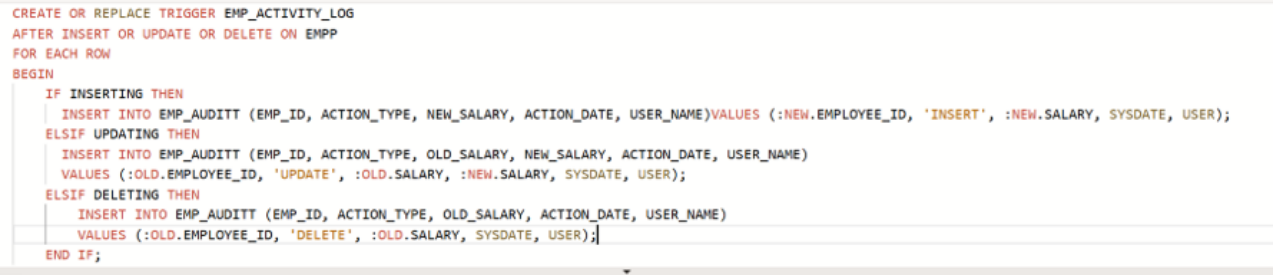
Write a code in PL/SQL to design a trigger that captures changes made to specific columns and logs them in an audit table.

```
CREATE OR REPLACE TRIGGER TR_EMP AFTER UPDATE OF SALARY ON EMPP FOR EACH ROW
BEGIN
IF :OLD.SALARY!=:NEW.SALARY THEN
INSERT INTO EMP_AUDIT VALUES(:OLD.EMPLOYEE_ID,'SALARY',TO_CHAR(:OLD.SALARY),TO_CHAR(:NEW.SALARY),'USER',SYSDATE);
END IF;
END;
/
```

Program 5

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 EMP_ACTIVITY_LOG
AFTER INSERT OR UPDATE OR DELETE ON EMPP
FOR EACH ROW
BEGIN
    IF INSERTING THEN
      INSERT INTO EMP_AUDITT (EMP_ID, ACTION_TYPE, NEW_SALARY, ACTION_DATE, USER_NAME)VALUES (:NEW.EMPLOYEE_ID, 'INSERT', :NEW.SALARY, SYSDATE, USER);
    ELSIF UPDATING THEN
      INSERT INTO EMP_AUDITT (EMP_ID, ACTION_TYPE, OLD_SALARY, NEW_SALARY, ACTION_DATE, USER_NAME)
      VALUES (:OLD.EMPLOYEE_ID, 'UPDATE', :OLD.SALARY, :NEW.SALARY, SYSDATE, USER);
    ELSIF DELETING THEN
        INSERT INTO EMP_AUDITT (EMP_ID, ACTION_TYPE, OLD_SALARY, ACTION_DATE, USER_NAME)
        VALUES (:OLD.EMPLOYEE_ID, 'DELETE', :OLD.SALARY, SYSDATE, USER);
    END IF;
```

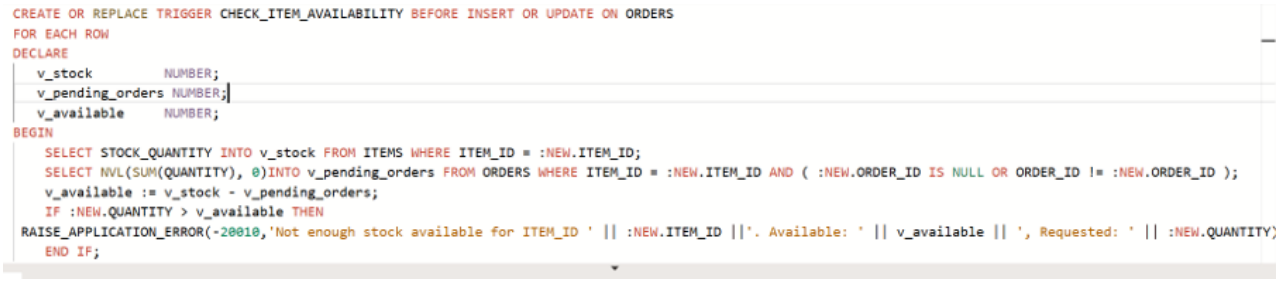
Program 7

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.

```
DECLARE
V_TOT NUMBER;
BEGIN
SELECT NVL(SUM(SALARY),0) INTO V_TOT FROM EMPP;
V_TOT := V_TOT + :NEW.SALARY;
END;
/
```

Program 8

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



```
CREATE OR REPLACE TRIGGER CHECK_ITEM_AVAILABILITY BEFORE INSERT OR UPDATE ON ORDERS
FOR EACH ROW
DECLARE
   v_stock         NUMBER;
   v_pending_orders NUMBER;
   v_available     NUMBER;
BEGIN
    SELECT STOCK_QUANTITY INTO v_stock FROM ITEMS WHERE ITEM_ID = :NEW.ITEM_ID;
    SELECT NVL(SUM(QUANTITY), 0)INTO v_pending_orders FROM ORDERS WHERE ITEM_ID = :NEW.ITEM_ID AND ( :NEW.ORDER_ID IS NULL OR ORDER_ID != :NEW.ORDER_ID );
    v_available := v_stock - v_pending_orders;
    IF :NEW.QUANTITY > v_available THEN
RAISE_APPLICATION_ERROR(-20010,'Not enough stock available for ITEM_ID ' || :NEW.ITEM_ID ||'. Available: ' || v_available || ', Requested: ' || :NEW.QUANTITY)
    END IF;
```

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