SIDDARTHA INSTITUTE OF SCIENCE AND TECHNOLOGY (AUTONOMOUS)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Ananthapuramu)
(Accredited by NBA for EEE, Mech., ECE & CSE and Accredited by NAAC with „A" Grade)
Siddartha Nagar, Narayanavanam Road, Puttur-517583

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

DATABASE MANAGEMENT SYSTEMS LAB

(20CS0508)

II B.TECH -I SEMESTER

SIDDARTHA INSTITUTE OF SCIENCE AND TECHNOLOGY: PUTTUR

VISION OF THE INSTITUTION

To become an eminent academic institute for academic and research that producing global leaders in science and technology to serve the betterment of mankind.

MISSION OF THE INSTITUTION

M1. To provide broad-based education and contemporary knowledge by adopting modern teaching-learning methods.

M2. To inculcate a spirit of research and innovation in students through industrial interactions.

M3. To develop individual‟s potential to its fullest extent so that they can emerge as gifted leaders in their fields.

DEPARTMENT OF COMPUETR SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a well-known department of Computer Science and Engineering producing competent professionals with research and innovation skills, inculcating moral values and societal concerns.

MISSION OF THE DEPARTMENT

M1. To educate students to become highly qualified computer engineers with full commitments to professional ethics.

M2. To inculcate a mind of innovative research in the field of computer science and related interdisciplinary areas to provide advanced professional service to the society.

M3. To prepare students with industry ready knowledge base as well as entrepreneurial skills by introducing duly required industry oriented educational program.

PROGRAM EDUCATIONAL OBJECTIVES STATEMENTS

PEO1. Graduates with basic and advanced knowledge in science, mathematics, computer science and allied engineering, capable of analyzing, design and development of solutions for real life problems.

PEO2. Graduates who serve the Industry, consulting, government organizations, or who pursue higher education or research.

PEO3. Graduates with qualities of professional leadership, communication skills, team work, ethical values and lifelong learning abilities.

PROGRAMME OUTCOMES (PO'S):

PO1 **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2 **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3 **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 **odern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6 **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7 **vironment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8 **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9 **dividual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 **mmunication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11 **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‟s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12 **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAMME SPECIFIC OUTCOMES (PSO'S):

PSO1 **Architecture of Computer System**: Ability to visualize and articulate computer hardware and software systems for various complex applications.

PSO2 **Design and develop computer programs**: Ability to design and develop computer-based systems in the areas related to algorithms, networking, web design, cloud computing, IoT, data analytics and mobile applications of varying complexity.

PSO3 **Applications of Computing and Research Ability:** Ability to use knowledge in various domains to identify research gaps and hence to provide solution to new ideas and innovations.

SIDDARTHA INSTITUTE OF SCIENCE AND TECHNOLOGY: PUTTUR
(AUTONOMOUS)

II B.Tech. – I Sem.	**L**	**T**	**P**	**C**
	-	**-**	**3**	**1.5**

(20CS0508) DATABASE MANAGEMENT SYSTEMS LAB

COURSE OBJECTIVES

The objectives of this course

1. *Illustrate the different issues involved in the design and implementation of a database system.*
2. *Use data manipulation language to query, update, and manage a database.*
3. *Design and build a simple database system and demonstrate competence with the fundamental tasks involved with modeling, designing, and implementing a DBMS.*

COURSE OUTCOMES (COs)

On successful completion of this course, the student will be able to:

1. *Develop relational algebra expressions for queries and optimize them.*
2. *Design the databases using E_R method for a given specification of requirements.*
3. *Apply Normalization techniques on given database.*
4. *Determine the transaction atomicity, consistency, isolation, and durability for a given transaction-processing system.*
5. *Implement the isolation property, including locking, time stamping based on concurrency control and Serializability of scheduling.*
6. *Execute DDL, DML, DCL commands.*

LIST OF EXPERIMENTS:

1. Practice session: Students should be allowed to choose appropriate DBMS software, install it, configure it and start working on it. Create sample tables, execute some queries, use SQLPLUS features, and use PL/SQL features like cursors on sample database. Students should be permitted to practice appropriate User interface creation tool and Report generation tool.
2. DDL Commands – Table Creation, Altering the table structures, truncating a table and dropping a table.
3. DML Commands – Insert, Select Commands, update & delete Commands.
4. Create relationship between the databases – Nested Queries & Join Queries
5. Create a database and to set various possible constraints.
6. Views – Create a Virtual table based on the result set of an SQL statement.
7. Create PL/SQL functions to implement the stored procedures in SQL (Function and Procedures).
8. Write a PL/SQL program using For loop to insert ten rows into a database table.
9. Write Relational algebra queries for a given set relations.
10. Write a PL/SQL program to execute a number of queries in one block using single command.

TEXT BOOK:

1. Raghu Ramakrishnan, Johannes Gehrke, Jeff Derstadt, Scott Selikoff and Lin Zhu, *Database Management Systems solutions manual*, third Edition, 2013.

SIDDARTHA INSTITUTE OF SCIENCE AND TECHNOLOGY: PUTTUR
(AUTONOMOUS)

DEPARTMENT OF COMPUETR SCIENCE AND ENGINEERING

Do's

- Follow the right procedures for starting the computer and shutting down the computer to avoid loss of data and damage of computer components
- You should only be on the specific program/site your lab faculty has assigned for you.
- Report any problems/damages immediately to the lab In-charge.
- Always use a light touch on the keyboard.
- When working on documents always save several times while working on it.
- Remember to log out whenever you are done using any lab computer.
- Shut down computer properly before you leave the lab.
- Know the location of the fire extinguisher and the first aid box and how to use them in case of an emergency.
- Read and understand how to carry out an experiment thoroughly before coming to the laboratory.

Dont's

- Computers and peripherals are not to be moved or reconfigured without approval of Lab In-charge.
- Students may not install software on lab computers. If you have a question regarding specific software that you need to use, contact the Lab In-charge.
- Never bang on the keys.
- Do not open up the metallic covers of computers or peripherals devices a particularly when the power is on.
- Never open attachments from unknown sources.
- Keep all liquids away from computers and equipment, liquid may spill and cause an electrical shock or the computer not to operate properly.
- Do not insert metal objects such as clips, pins and needles into the computer casings. They may cause fire.
- Do not touch, connect or disconnect any plug or cable without Lab in-charge‟s permission.

EX.NO: 1	INSTALLATION OF ORACLE 10g SOFTWARE

AIM:

To choose appropriate DBMS software, install it, configure it and start working on it.

Procedure:

Oracle database 10g express edition installation

Welcome to Oracle Database 10g Express Edition (Oracle Database XE)! This utorial gets you quickly up and running using Oracle Database XE by creating a simple pplication. This guide covers the following topics:

- Logging in as the Database Administrator
- Unlocking the Sample User Account
- Logging in as the Sample User Account
- Creating a Simple Application
- Running Your New Application
- Using the Oracle Database XE Menus

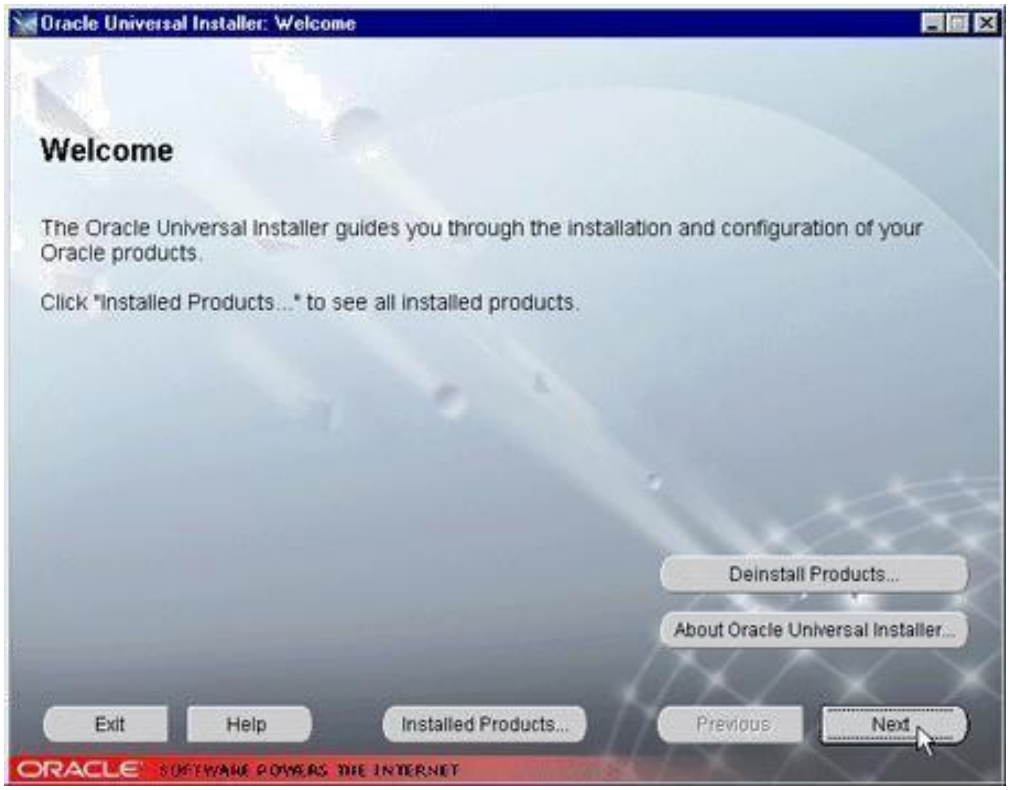
(A) Installing the Oracle10g Database

Perform the following steps to install the Oracle10g database:

1. The Oracle Universal Installer requires 400 MB of temporary space on your C: drive to run successfully. If you do not have at least 400 MB on the C: drive, then update the **TEMP** user variable in your environment variables and change it to a location with the required amount of space. In addition, make sure that the computer has sufficient system memory (the combination of RAM and virtual memory). If the page file size is too small, messages appear indicating that the computer is running low on virtual memory. Check your page file size to make sure that the initial size is **200 MB** and the maximum **size is 400 MB.**

2. Insert your **Oracle10g Enterprise Edition CD-ROM** into the CD-ROM drive. In the Autorun window that appears, choose **Install/Deinstall Products**. If you do not have your laptop set up for Autorun capability, run **autorun.exe** directly from the **AUTORUN** directory on your Oracle9*i* Enterprise Edition CD-ROM. You will install Oracle9*i* in its own home directory. Click **Next** to accept the default Oracle_Home name and location.

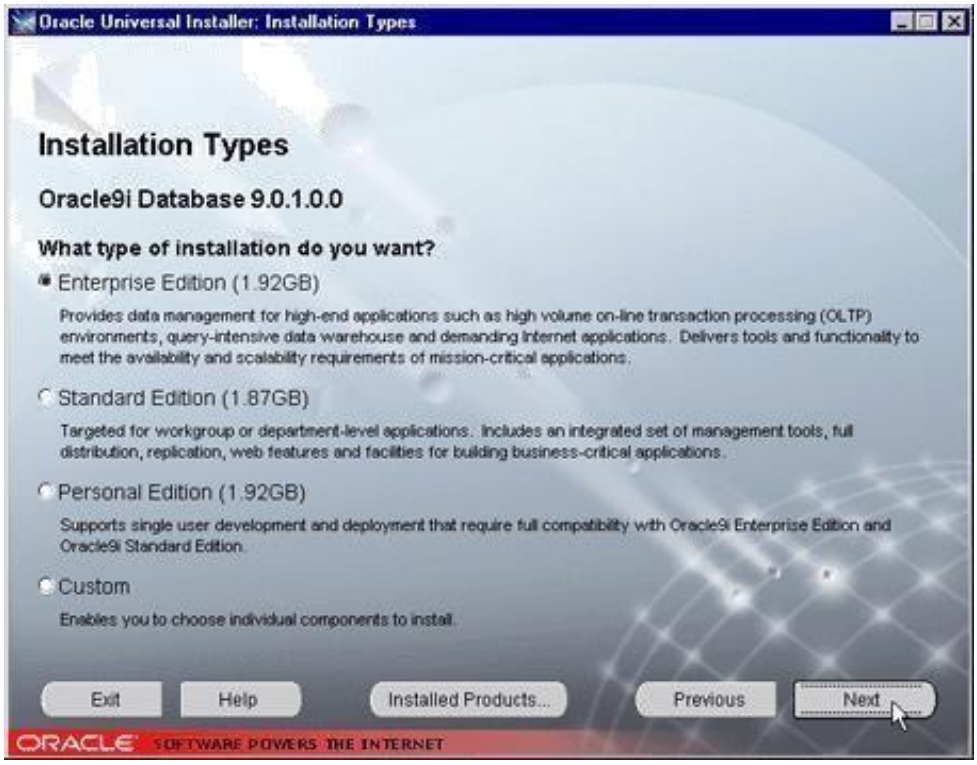
3. In the Welcome window, click **Next**.



4. Install Oracle10g in its own home directory. Make sure it says **OraHome90** in the Name field, and that the path is **d:\oracle\ora90**. Then click **Next**.

5. Select **Oracle9*i* Database 9.0.1.0.0**. Click **Next** to accept the default.

6. Choose **Enterprise Edition** and then click **Next**.



7. Accept the **General Purpose** default database configuration and click **Next**.

8. If a database already exists, you are asked whether you want to upgrade or migrate the database. Do not upgrade or migrate the database: click **Next**.

9. In the Global Database Name field, enter **orcl.world**. In the SID field, accept the default, **orcl**, and then click **Next**.

10. Change the directory to **D:\oracle\oradata** and click **Next**.

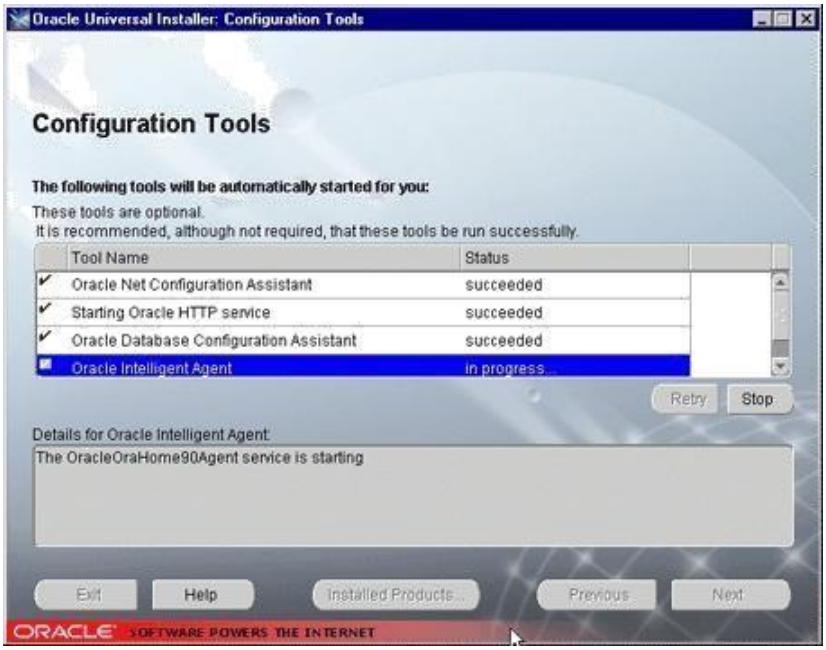
11. Accept the default character set and click **Next**.

12. The Oracle Universal Installer displays a summary of the installation options. Make sure that you have the required disk space available and then click **Install**.

13. The Oracle Universal Installer begins installing Oracle9*i* Enterprise Edition and related software. **Note:** This process takes at least one hour. The Oracle Universal Installer also creates installation logs as it progresses. You can view these while the installation is in progress. The log for this installation session is located at

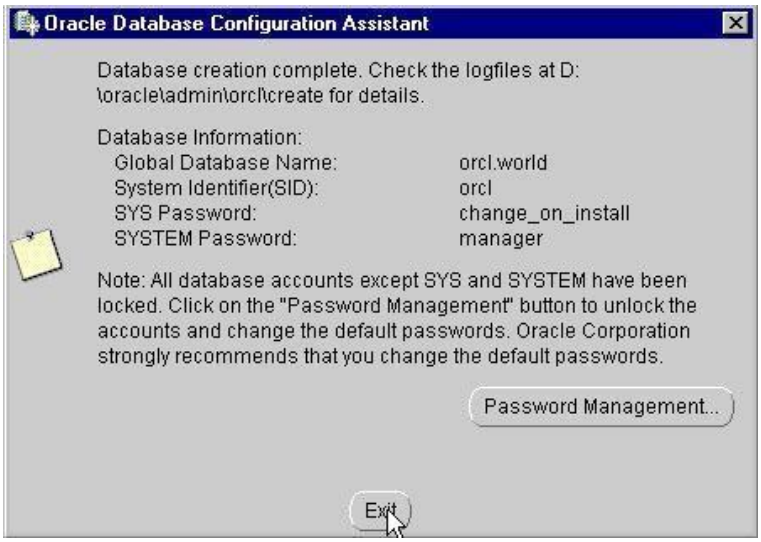
14. C:\ProgramFiles\Oracle\Inventory\logs\installActions.log. If you install another product, the current log is copied to this file and renamed, so there is always a history of each successive installation.

15. After the initial software is copied to the disk, the Oracle Universal Installer also automatically configures the network for the Production database. This is done in the background. The success of the operation is reported in the Oracle Universal Installer Configuration Tools window. Once the network configuration has been completed, the Oracle Universal Installer starts the Oracle Database Configuration Assistant. This assistant installs and configures the initial database in the background as well.



16. Database creation takes about twenty minutes. The progress is shown in the Database Creation Progress window. When the database has been created, the Database NT service is created.

17. Click **Exit**.



18. When the database has been completely created and all the configuration tools have completed their tasks, the End of Installation window appears. Click **Exit** and then **Yes** to exit the Universal Installer.

Testing the Oracle10g Installation

In this section you will test the installation of the Oracle9*i* database using Enterprise Manager, which combines all of the management tools together into one product.

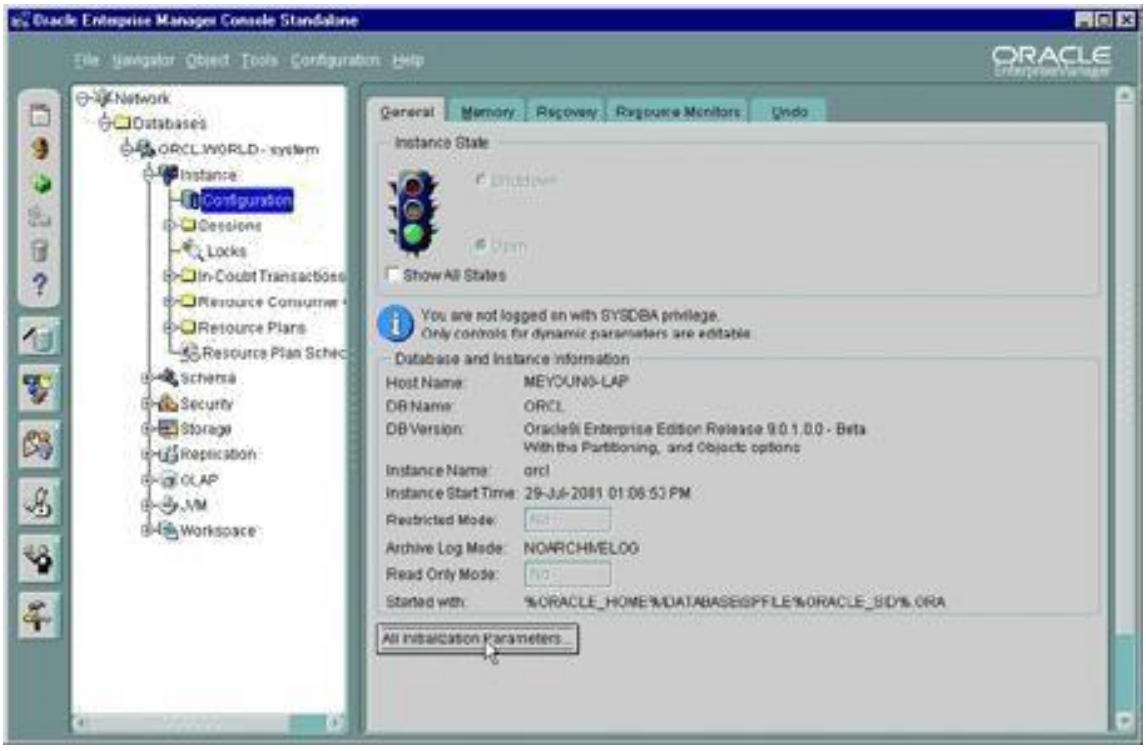
1. Select **Start > Programs > Oracle-OraHome90 > Enterprise Manager Console**.

2. In the Oracle Enterprise Manager Console login window, make sure that **Launch standalone** is selected and then click **OK**. Note: You can launch Enterprise Manager to immediately connect to the database for direct database administration. No configuring is necessary. From this Login window, you can also connect to the Oracle Management Server (OMS) however you need to create an OMS repository first which you will do in another module.

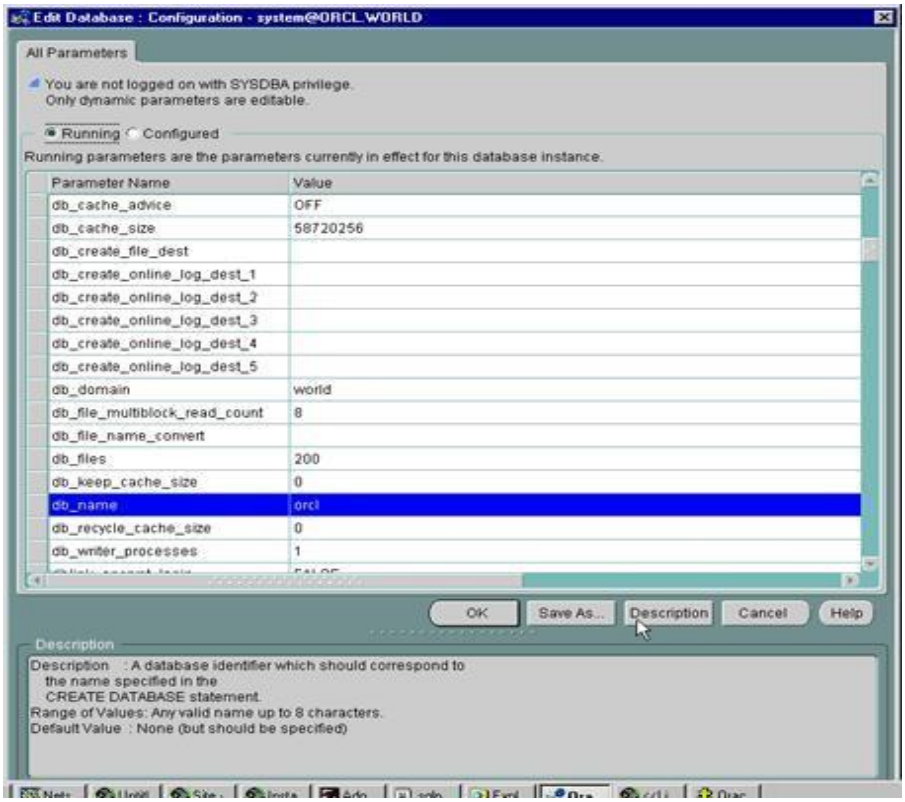
3. Expand **Network** then **Databases,** then **click the plus sign** to expand **ORCL.WORLD**

4. To log in to the instance, enter **system** in the Username field and **manager** in the Password field, then click **OK**.

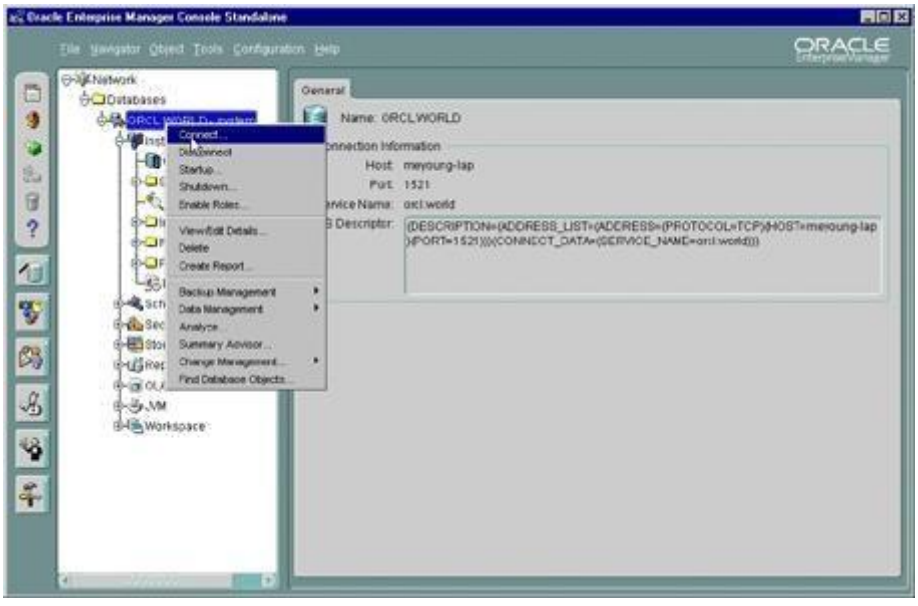
5. Expand **Instance** and then select **Configuration** to see the state of the database. To view the initialization parameters, click the **All Initialization Parameters** button.



6. Click the **db_name** parameter and then click the **Description** button. A description of the parameter is displayed at the bottom of the window. Click **OK** to close the window.

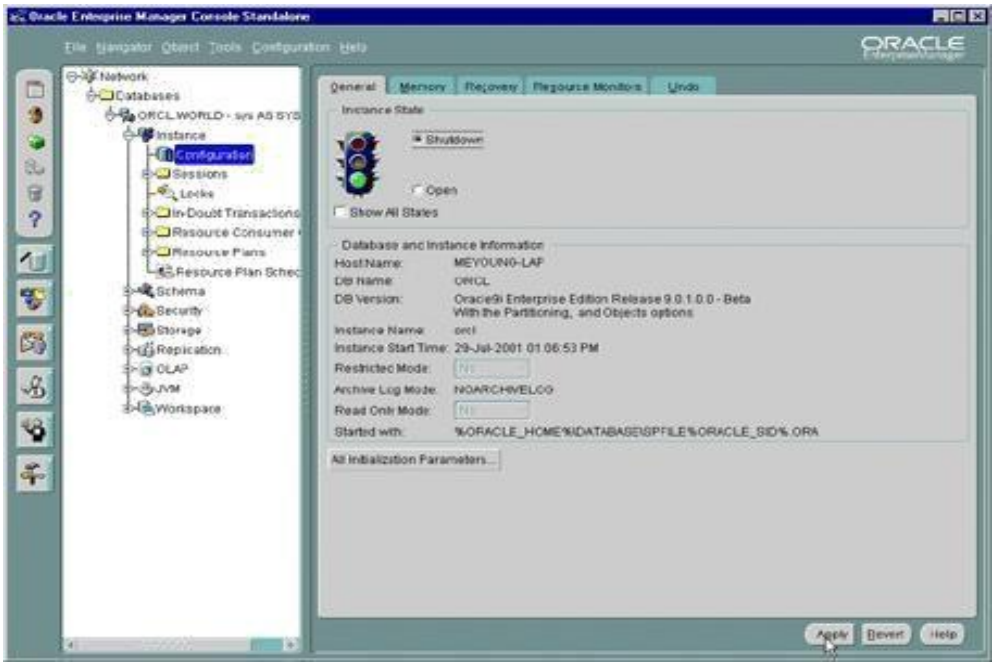


7. Notice that you cannot start or stop the database unless you are connected as SYSDBA. Right-click the **ORCL.WORLD** database and select **Connect**.

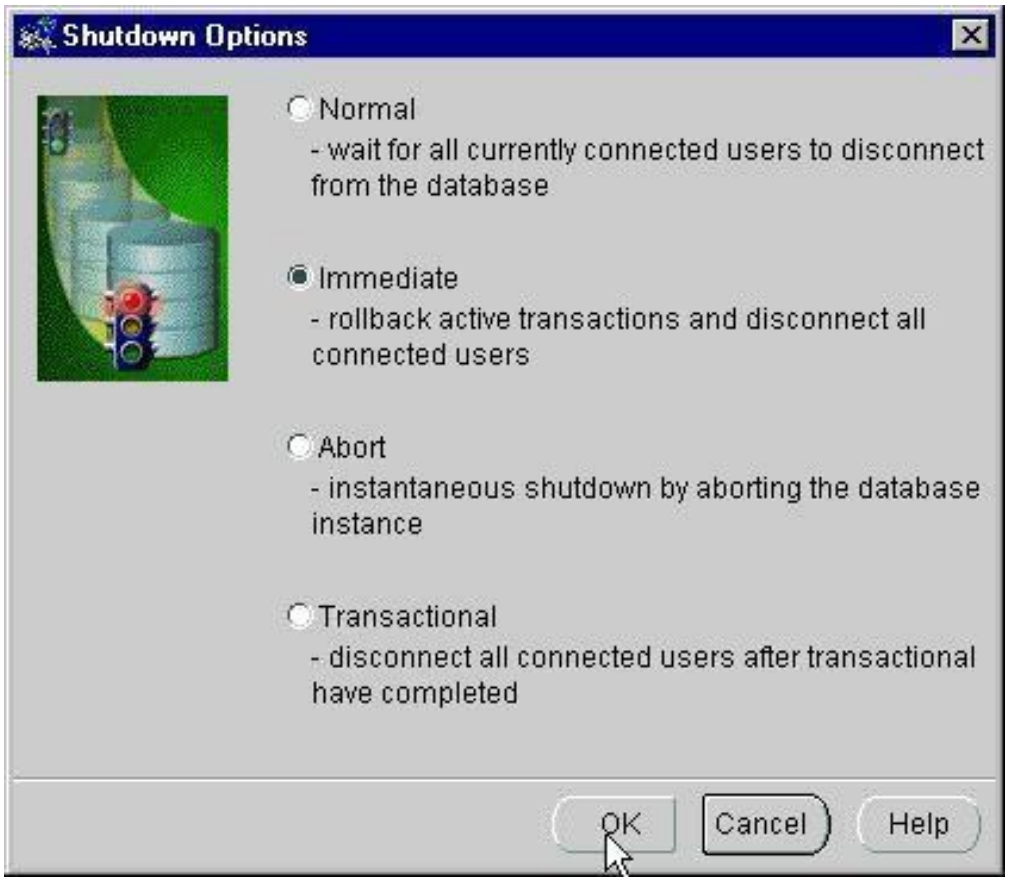


8. Enter **sys** in the **Username** field and **change_on_install** in the Password field. Select **SYSDBA** from the **Connect as** drop-down list and then click **OK**.

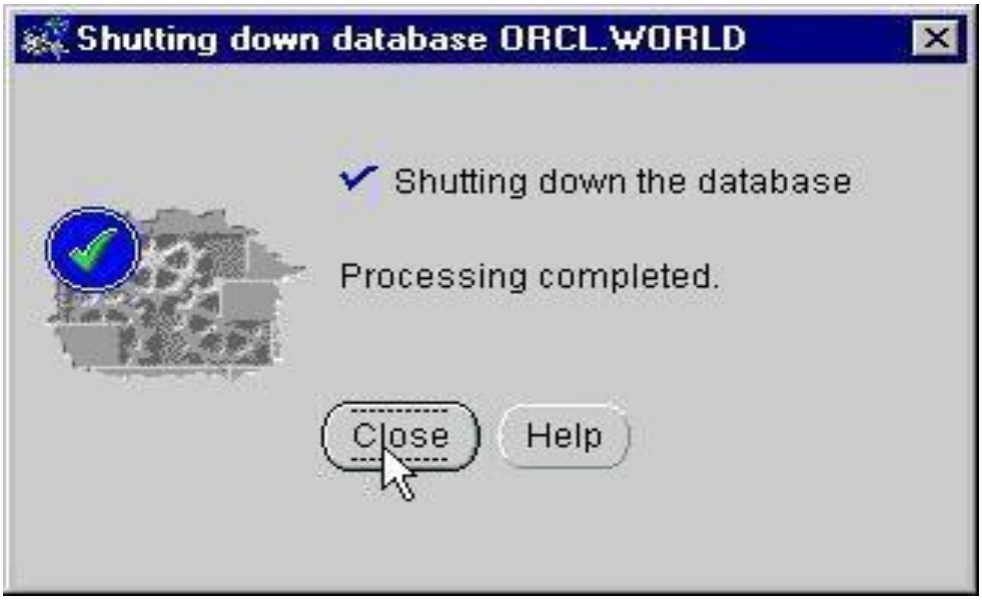
9. Expand **Instance** and then select **Configuration**. Notice that you now can shut down the database. Select the **Shutdown** option button and click **Apply**.



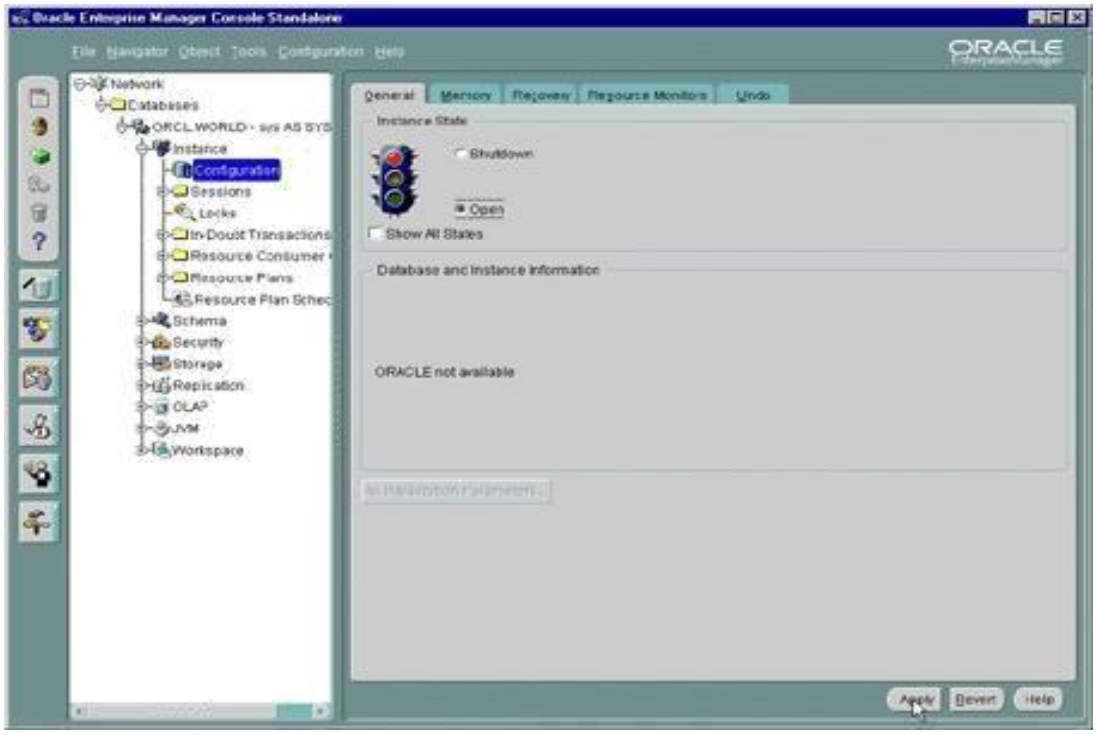
10. Click **OK** to accept the default choice and shut down immediately.



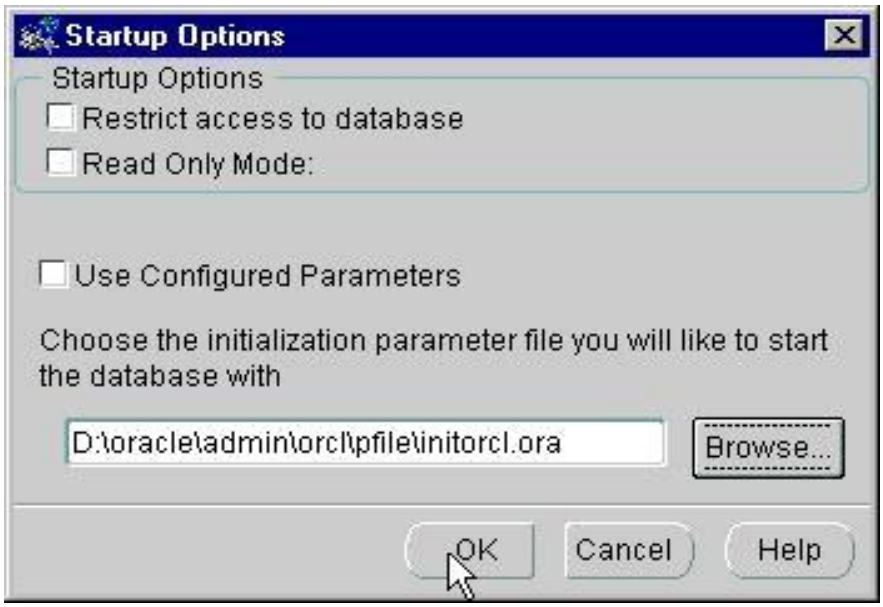
11. When the shutdown is complete, click **Close**.



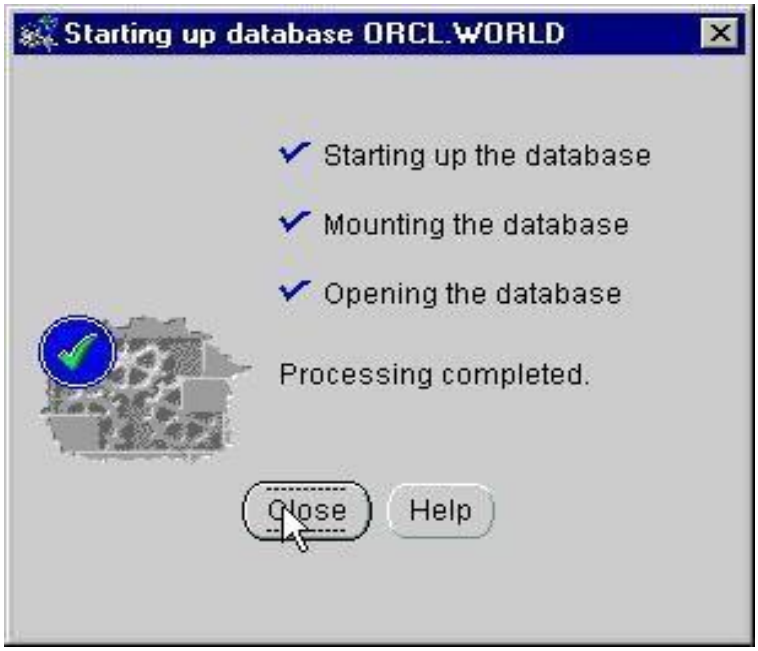
12. Notice that the database is no longer available. Now start it again. Click the **Open** option button and then **Apply**.



13. Clear the **Use Configured Parameters** check box. Click **Browse** to find the **D:\Oracle\admin\orcl\pfile\initorcl.ora** file and then click **OK**.

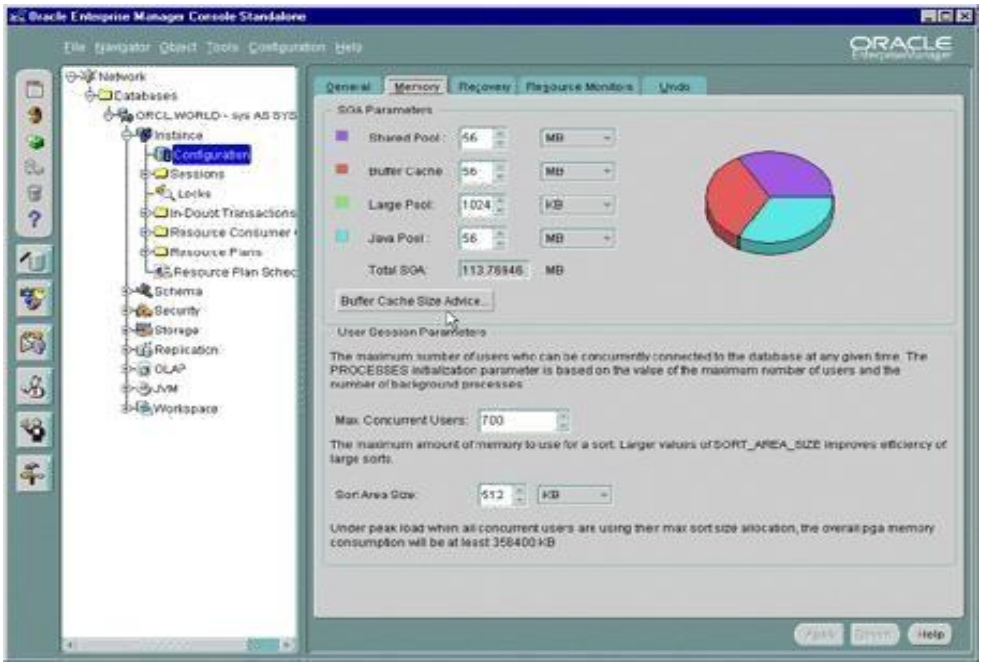


14. When the database has been successfully started up, click **Close**.



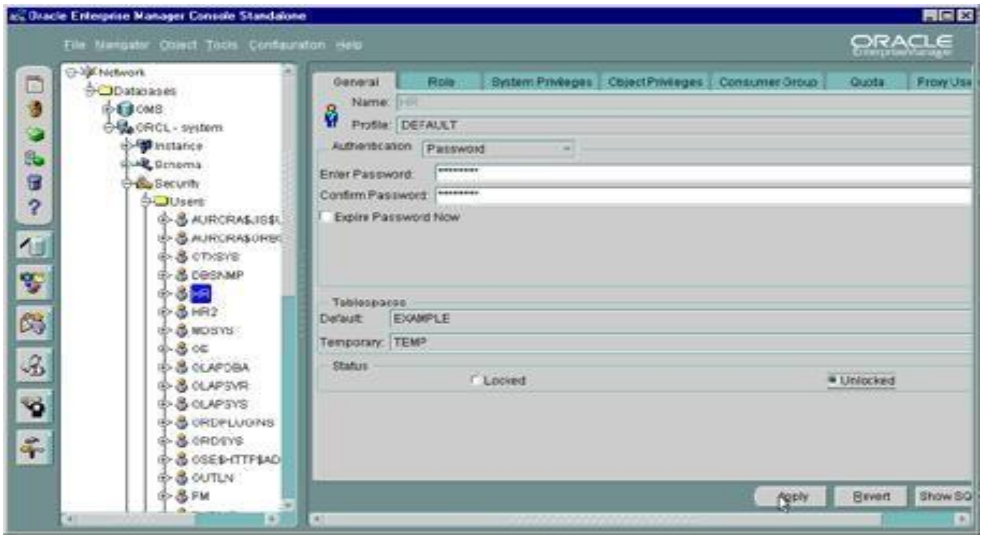
15. Notice that the database has been started. Click the **Memory** tab.

16. In the Memory tabbed page, you can see how memory is currently allocated.



17. When you created your database, the sample schemas (or seeded) are loaded into your database, the HR user cannot log into the database until the password has been changed or that user is unlocked. This lock has been put into place for security purposes. You can unlock the user from Enterprise Manager by expanding **Security**, then **Users**, then selecting **HR**.

18. Select **Unlock** then **Apply**. Unlock the other Sample Schema Users: OE, PM, SH, and QS.



19. Select **File > Exit** to close the Enterprise Manager Console.

EX.NO: 2	DDL COMMANDS

AIM:

To write a query to implement DDL commands such as create, alter, truncate and drop.

PROCEDURE:

DDL - Data Definition Language

S.No	Command & Description
1	CREATE - Creates a new table, a view of a table, or other object in the database.
2	ALTER - Modifies an existing database object, such as a table.
3	DROP - Deletes an entire table, a view of a table or other objects in the database.
4	TRUNCATE- This command used to delete all the rows from the table and free the space containing the table.

1. CREATE A TABLE

Syntax:

create table <tablename>(columnname1 datatype1(size)..... columnnameN datatypeN(size));

Example:

SQL>create table employee(eno number(3),ename char(10),sal number(7,2));

Output:

SQL>desc employee;

Name	Null?	Type
ENO		NUMBER(3)
ENAME		CHAR(10)
SAL		NUMBER(7,2)

2.Alter

add
modify
drop
rename

Add

Syntax

:

alter table <tablename> add(newcolumnnewdatatype(size)..... N);

Example:

alter table employee add(address char(20),phno number(10));

Output:
Table altered.
SQL>desc employee;

Name	Null?	Type
ENO		NUMBER(3)
ENAME		CHAR(10)
SAL		NUMBER(7,2)
ADDRESS		CHAR(20)
PHNO		NUMBER(10)

Modify

Syntax:

alter table <tablename> modify(oldcolumnnamedatatype(new size));

Example:

alter table employee modify(eno number(4));

Output:
Table altered.
SQL>desc employee;

Name	Null?	Type
ENO		NUMBER(4)
ENAME		CHAR(10)
SAL		NUMBER(7,2)
ADDRESS		CHAR(20)
PHNO		NUMBER(10)

Drop

Syntax:

alter table <tablename> drop column oldcolumnname;

Example:

```
alter table employee drop column address;
```

Output:
Table altered.

SQL>desc employee;

Name	Null?	Type
EnoNUMBER(4)		
ENAME		CHAR(10)
SAL		NUMBER(7,2)
PHNO		NUMBER(10)

Rename

Syntax:

alter table <tablename> rename column oldcolumn to newcolumn;

Example:

alter table employee rename column eno to emp_no;

Output:

Table altered.

SQL>desc employee;

Name	Null?	Type
EMP_NO		NUMBER(4)
ENAME		CHAR(10)
SAL		NUMBER(7,2)
PHNO		NUMBER(10)

3.Drop

Syntax:

drop table <tablename>;

Example:

drop table employee;

Output:

Table dropped.

SQL>desc employee;

ERROR:
ORA-04043: object employee does not exist

4. Truncate:

Syntax:

truncate table <tablename>;

Example:

truncate table employee;

Output:

Table truncated.

SQL> select *from employee;

Output:
no rows selected

SQL>desc employee;

Name	Null?	Type
EMP_NO		NUMBER(4)
ENAME		CHAR(10)
SAL		NUMBER(7,2)
PHNO		NUMBER(10)

EX.NO: 3	DML COMMANDS

AIM: To write a query to implement DML commands such as insert, update, delete and select.

PROCEDURE:

DML - Data Manipulation Language

SNO	Command & Description
1	SELECT - Retrieves certain records from one or more tables.
2	INSERT - Creates a record.
3	UPDATE - Modifies records.
4	DELETE - Deletes records.

DML COMMANDS(Data Manipulation Commands)

create table employee(eno number(10), ename varchar2(20), sal number(5));

1. Insert

Syntax:

insert into <tablename> values(coumnname1 values1.....N);

Example:

insert into employee values(101,'king',1000);

output:

1 row created

Dynamic values

insert into employee values(&eno,'&ename',&sal);

output:

Enter value for eno: 102
Enter value for ename: arjun
Enter value for sal: 2000
old 1: insert into employee values(&eno,'&ename',&sal)
new 1: insert into employee values(102,'arjun',2000)
1 row created.
SQL> /
Enter value for eno: 103
Enter value for ename: thanuja
Enter value for sal: 5000
old 1: insert into employee values(&eno,'&ename',&sal)
new 1: insert into employee values(103,'thanuja',5000)
1 row created.

Selected column names

insert into employee (eno) values(101);

Output:

1 row created.

insert into employee (ename,sal) values('&ename',&sal);

Output:

Enter value for ename: varun
Enter value for sal: 4000
old 1: insert into employee (ename,sal) values('&ename',&sal)
new 1: insert into employee (ename,sal) values('varun',4000)
1 row created.
SQL> select *from employee;

OUTPUT:

ENO	ENAME	SAL
101	king	1000
102	arjun	2000
103	thanuja	5000
105	varun	4000

2. Update

Syntax:

update<tablename> set oldcomumn=newvalue where condition;

Example:

update employee set sal=10000 where eno=101;

Output:

1 row updated.
SQL> select *from employee;

ENO	ENAME	SAL
101	king	10000
102	arjun	2000
103	thanuja	5000
105	varun	4000

3. Delete

Syntax:

delete from <tablename> where condition;

Example:

Delete from employee where eno=102;

Output:

1 row deleted.
SQL> select *from employee;

ENO	ENAME	SAL
101	king	10000
103	thanuja	5000
105	varun	4000

Data Query Language:

Select

Syntax:

select * from <tablename>;

select * from <table name> where condition;

Example:

Select * from employee;

Output:

EMPNO	ENAME	MGRSSN	SALARY	DEPTNO	JOB
111	nirmala	3456	40000	2	lecturer
112	nandhu	5678	80000	1	surgeon
113	sathya	2345	90000	3	marketing
114	vino	6789	50000	2	doctor

EX.NO: 4a	NESTED QUERIES

AIM:

To write and implement the SQL queries for nested queries

PROCEDURE:

Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow −

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows −

```
SELECT column_name [, column_name ]FROM table1 [, table2 ]
 WHERE column_name OPERATOR (SELECT column_name [, column_name ] FROM
   table1 [, table2 ][WHERE])
```

Consider the CUSTOMERS table having the following records

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Now, let us check the following subquery with a SELECT statement.

```
SQL> SELECT * FROM CUSTOMERS WHERE ID IN
        (SELECT ID FROM CUSTOMERS WHERE
        SALARY > 4500) ;
```

This would produce the following result.

OUTPUT:

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Subqueries with the INSERT Statement

The basic syntax is as follows.

```
INSERT INTO table_name [(column1 [, column2 ]) ]
SELECT [*|column1 [,column2 ] FROM table1 [,table2 ][WHERE VALUE OPERATOR ]
```

Example

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

```
SQL> INSERT INTO CUSTOMERS_BKP SELECT * FROMCUSTOMERS
    WHERE ID IN (SELECT ID FROM CUSTOMERS) ;
```

OUTPUT:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Subqueries with the UPDATEStatement

The basic syntax is as follows.

```
UPDATE table SET column_name = new_value [ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME FROM TABLE_NAME)[ WHERE)]
```

Example

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
SQL> UPDATE CUSTOMERS SET SALARY = SALARY * 0.25
      WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP WHERE AGE >= 27 );
```

This would impact two rows and finally CUSTOMERS table would have the following records.

OUTPUT:

Two rows updated

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	500
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	2125.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Subqueries with the DELETEStatement

The basic syntax is as follows.

```
DELETE FROM TABLE_NAME [ WHERE OPERATOR [VALUE ] (SELECT
COLUMN_NAME FROM TABLE_NAME)[ WHERE) ]
```

Example

Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
SQL> DELETE FROM CUSTOMERS WHERE AGE IN (SELECT AGE FROM
     CUSTOMERS_BKP WHERE AGE >= 27 );
```

OUTPUT:

Two rows updated

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

NESTED QUERIES OR SUB QUERIES:

Syntax:

Select<column(s)>from table where<condn operator> (select <column>from table);

Query: select * from employee;

Output:

SSN	NAME	BDATE	SALARY	MGRSSN	DNO
1111	sathya	17-DEC-88	50000	4323	3
1112	vinotha	02-AUG-90	32000	5462	1
1113	nandu	07-OCT-93	30000	6452	2
1114	rajesh	05-APR-85	40000	8264	2
1115	nive	06-OCT-92	45000	7241	1

Query: select * from department;

Output:

DNO	DNAME	LOC
1	admin	Madurai
2	research	Chennai
3	accounts	Bangalore

Q1: Display the names of the employees working for Accounts department.

Query:

SQL> select name from employee where dno=(select dno from department where dname='accounts');

Output:

NAME
sathya

Q2: Display names of employees whose salary is greater than the employee SSN=1234

Query:

SQL> select name from employee where salary>(select salary from employee wheressn=1234)

Output:

NO ROWS SELECTD

Q3: Display all the employees drawing more than or equal to the average salary of department number 3.

Query:

SQL> select name from employee where salary>=(select avg(salary) from employee group by dno having dno=3);

Output:

NAME
Sathya

Q4: Display the name of the highest paid employee.

Query: SQL> select name from employee where salary=(select max(salary) from employee);

Output:

NAME
Sathya

Q5: Find the Name and Salary of people who draw in the range Rs. 20,000 to Rs. 40,000.

Query:

SQL> select name, salary from employee where salary in(select salary from employee where salary between 20000 and 40000);

Output:

NAME	SALARY
vinotha	32000.00
nadu	30000
Rajesh	40000

EX.NO: 4b	JOIN QUERIES

AIM: To write a query to implement the Join using SQL queries.

SQL JOIN:

Procedure:

- SQL JOINS are the special clauses that are used to combine multiple tables present in a database on the basis of some common attributes present in those tables.
- The SQL JOINS has the ability of combining two or more data tables/tuples into a single table/table only if the following conditions are satisfied.
  - There must be a common attribute in both(tables which are participating) tables.
  - Join condition must be satisfied.
- The common attributes are compared using SQL Operators based upon the required conditions. The most oftenly used operator is the "=" (Equal To) symbol.
- The concept of joins used in SQL are similar to that in DBMS except for the syntax. Also, SQL Joins uses "ON" clause. There are Five basic SQL Joins each of whose syntax and examples are explained in this chapter.

SQL Joins : Types

SQL JOINS : The INNER Join

- The INNER JOIN which is also known as "SIMPLE JOIN or EQUIJOIN" is the most commonly used join in SQL. Queries equipped with INNER JOIN when executed returns all the rows which are present in the tables corresponding to the common attribute.

Syntax :

SQL> SELECT Column_Name From Table_Name1 INNER JOIN Table_Name2 ON Table_Name1.Column_Name = Table_Name2.Column_Name;

For Example : Consider two tables student_details and student_result.

Table : student_details

Roll_No	Name	Address
1.	Anoop	Delhi
2.	Anurag	Noida
3.	Rakesh	Ghaziabad
4.	Saurav	Faridabad

Table : student_result

Roll_No	Marks
1.	50
2.	60
3.	70

Query :

SQL> SELECT * From student_details INNER JOIN student_result Where student_details.Roll_No = student_result.Roll_No;

Output :

Roll_No	Name	Address	Roll_No	Marks
1.	Anoop	Delhi	1.	50
2.	Anurag	Noida	2.	60
3.	Rakesh	Ghaziabad	3.	70

Duplicate Column

SQL JOINS : The NATURAL Join

- The NATURAL JOIN is similar to INNER JOIN. The only difference is, redundant columns present in INNER JOINS are removed when NATURAL JOIN is used.
- It works only if, one attribute or more than one attributes are common between the joining/participating relations.

Syntax : SELECT Column_Name From Table_Name1 NATURAL JOIN Table_Name2;

Example : Consider two tables student_details and student_result.
Query :

SQL>SELECT * From student_details NATURAL JOIN student_result;
Output

Student_Details

Roll No.	Name	Address
1	Anoop	Delhi
2	Anurag	Noida
3	Ganesh	UP

⋈

Student_Result

Roll No.	Marks
1	20
2	30
3	10

Student_Details ⋈ Student_Result

Roll No.	Name	Address	Marks
1	Anoop	Delhi	20
2	Anurag	Noida	30
3	Ganesh	UP	10

SQL JOINS : LEFT OUTER JOIN

- There exists a concept of position(left or right) of relations in case of both LEFT and RIGHT OUTER JOIN.
- To implement LEFT OUTER JOIN, at least one entity needs to be common in the relations. All the attributes/tuples present in the left relation are recorded in the resulting relation along with those which are commonly present in the right relation.
- If in case any tuple in left relation does not matches with the tuple in right relation, NULL value will be displayed against that tuple in the resulting relation.

Syntax : SELECT Column_Name From Table_Name1 LEFT JOIN Table_Name2 ON Table_Name1.Column_Name = Table_Name2.Column_Name;

Example : Consider the tables student_details and student_result. Now, if we want to implement left outer join on these relations, the result will look like:

Query :

SQL>SELECT Roll_No, Name, Address, Subject, Marks From Student_Details LEFT OUTER JOIN Student_Result ON Student_Details.Roll_No = Student_Result.Roll_No;

Student_Details

Roll No.	Name	Address
1	Anoop	Delhi
2	Anurag	Noida
3	Ganesh	Ghaziabad
4	Saurav	Faridabad

⋈

Student_Result

Roll No.	Subject	Marks
1	DBMS	10
2	Java	20
3	C++	25

Student_Details ⋈ Student_Result

Roll No.	Name	Address	Roll No.	Subject	Marks
1	Anoop	Delhi	1	DBMS	10
2	Anurag	Noida	2	Java	20
3	Ganesh	Ghaziabad	3	C++	25
4	Saurav	Faridabad	Null	Null	Null

SQL JOINS : The RIGHT OUTER Join

- There exists a concept of position(left or right) of relations in case of both LEFT and RIGHT OUTER JOIN.
- The RIGHT OUTER JOIN is completely similar to left outer join except the resulting relation will include all the tuples from relation present on right hand relation.
- Also, NULL value will be displayed against the tuple which doesn‟t matches up with the left side relation.

Syntax : SELECT Column_Name From Table_Name1 RIGHT JOIN Table_Name2 ON

Table_Name1.Column_Name = Table_Name2.Column_Name;

Example : Consider the tables Student_Details and Student_Result. Now, if we want to implement right outer join on these relations, the result will look like:

Query :
SQL>SELECT Roll_No, Name, Address, Subject, Marks From Student_Details RIGHT OUTER JOIN Student_Result ON Student_Details.Roll_No = Student_Result.Roll_No;

Student_Details

Roll No.	Name	Address
1	Anoop	Delhi
2	Anurag	Noida
3	Ganesh	Ghaziabad
4	Saurav	Faridabad

⟖

Student_Result

Roll No.	Subject	Marks
1	DBMS	10
2	Java	20
3	C++	25
5	DLD	30

Student_Details ⟖ Student_Result

Roll No.	Name	Address	Roll No.	Subject	Marks
1	Anoop	Delhi	1	DBMS	10
2	Anurag	Noida	2	Java	20
3	Ganesh	Ghaziabad	3	C++	25
Null	Null	Null	5	DLD	30

SQL JOINS : The FULL OUTER Join

- In FULL OUTER JOIN, both the relations are merged together which results in a relation consisting of all the tuples.
- If in case, tuples doesn‟t matches, NULL value is passes against that.

Syntax : SELECT Column_Name From Table_Name1 FULL OUTER JOIN Table_Name2 ON Table_Name1.Column_Name = Table_Name2.Column_Name;

For example : Consider the tables Student_Details and Student_Result. Now, if we want to implement full outer join on these relations, the result will look like:

Query : SELECT Roll_No, Name, Address, Subject, Marks From Student_Details FULL OUTER JOIN Student_Result ON Student_Details.Roll_No = Student_Result.Roll_No;

Student_Details

Roll No.	Name	Address
1	Anoop	Delhi
2	Anurag	Noida
3	Ganesh	Ghaziabad
4	Saurav	Faridabad

⟗

Student_Result

Roll No.	Subject	Marks
1	DBMS	10
2	Java	20
3	C++	25
5	DLD	30

Student_Details ⟗ Student_Result

Roll No.	Name	Address	Roll No.	Subject	Marks
1	Anoop	Delhi	1	DBMS	10
2	Anurag	Noida	2	Java	20
3	Ganesh	Ghaziabad	3	C++	25
4	Saurav	Faridabad	Null	Null	Null
Null	Null	Null	5	DLD	30

NOTE : In the above description of various joins, some symbols are used in the screenshots of examples. They all have some certain meaning.

1. ⋈: Natural Join
2. ⟕ : Left Outer Join
3. ⟖: Right Outer Join
4. ⟗ : Full Outer Join

EX.NO: 5	CONSTRAINTS

AIM: To Practice and implement the Constraints using SQL queries.

PROCEDURE:

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

Following are some of the most commonly used constraints available in SQL.

1. NOT NULL Constraint − Ensures that a column cannot have NULL value.
2. DEFAULT Constraint − Provides a default value for a column when none is specified.
3. UNIQUE Constraint − Ensures that all values in a column are different.
4. PRIMARY Key − Uniquely identifies each row/record in a database table.
5. FOREIGN Key − Uniquely identifies a row/record in any of the given database table.
6. CHECK Constraint − The CHECK constraint ensures that all the values in a column satisfies certain conditions.
7. INDEX − Used to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

1.**NOT NULL Constraint** − Ensures that a column cannot have NULL value.

SQL NOT NULL on CREATE TABLE

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

Example:

Create table stud (sname varchar2(20) not null, rollno number(10) not null,dob date not null);

- CREATE TABLE STUDENT(ROLL_NO INT NOT NULL,STU_NAME VARCHAR(35) NOT NULL,STU_AGE INT NOT NULL,STU_ADDRESS VARCHAR(235),PRIMARY KEY (ROLL_NO));

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created.

Example:

- CREATE TABLE Persons (ID int NOT NULL,LastName varchar(255) NOT NULL,FirstName varchar(255) NOT NULL,Age int);

SQL NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

- ALTER TABLE Persons MODIFY Age int NOT NULL;

Q1: Add the constraint UNIQUE for regnumber attribute from studmarks table

Query: SQL> alter table studmarks add constraint s unique(regnumber);

Q2: Remove the constraint for the regnumber attribute

Query: SQL> alter table studmarks drop constraint s;

Q3: Add foreign key constraint for the column rollno from studmarks that refers rollno from student table.

Query: SQL> alter table studmarks add foreign key(rollno) references student(rollno);

Q4: Add one more column age in student table with NOT NULL constraint in student table

Query: SQL> alter table student add(age number(2) not null);

SQL> desc student;

DEFAULT Constraint − Provides a default value for a column when none is specified.

•The DEFAULT constraint is used to provide a default value for a column.

•The default value will be added to all new records IF no other value is specified.

- CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL,FirstName varchar(255), Age int,City varchar(255) DEFAULT 'Sandnes');

SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

- ALTER TABLE Persons MODIFY City DEFAULT 'Sandnes';
- DROP a DEFAULT Constraint
- ALTER TABLE Persons ALTER COLUMN City DROP DEFAULT;

UNIQUE Constraint − Ensures that all values in a column are different.

- CREATE TABLE Persons (ID int NOT NULL UNIQUE,LastName varchar(255) NOT NULL,FirstName varchar(255), Age int);
- CREATE TABLE STUDENT(ROLL_NO INT NOT NULL,STU_NAME VARCHAR (35) NOT NULL UNIQUE,STU_AGE INT NOT NULL,STU_ADDRESS VARCHAR (35) UNIQUE,PRIMARY KEY (ROLL_NO));

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

- CREATE TABLE Persons (ID int NOT NULL,LastName varchar(255) NOT NULL,FirstName varchar(255), Age int,CONSTRAINT UC_Person UNIQUE (ID,LastName));

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

ALTER TABLE Persons DROP CONSTRAINT UC_Person;

OUTPUT:

CONSTRAINT UC dropped

4. **PRIMARY Key** − Uniquely identifies each row/record in a database table.

- CREATE TABLE Persons (ID int NOT NULL PRIMARY KEY,LastName varchar(255) NOT NULL, FirstName varchar(255), Age int);

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

- CREATE TABLE Persons (ID int NOT NULL,LastName varchar(255) NOT NULL,FirstName varchar(255),Age int,CONSTRAINT PK_Person PRIMARY KEY (ID,LastName));

SQL PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

- ALTER TABLE Persons ADD PRIMARY KEY (ID);

- DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

- ALTER TABLE Persons DROP CONSTRAINT PK_Person;

5. **FOREIGN Key** − Uniquely identifies a row/record in any of the given database table. SQL

FOREIGN KEY Constraint

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

"Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

- The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
- The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into

the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

- CREATE TABLE Orders (OrderID int NOT NULL PRIMARY KEY, OrderNumber int NOT NULL,PersonID int FOREIGN KEY REFERENCES Persons(PersonID));

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

- CREATE TABLE Orders (OrderID int NOT NULL,OrderNumber int NOT NULL, PersonID int,PRIMARY KEY (OrderID),CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));

SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

- ALTER TABLE Orders ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

- ALTER TABLE Orders ADD CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

- SQL Server / Oracle / MS Access:

ALTER TABLE Orders DROP CONSTRAINT FK_PersonOrder;

6. **CHECK Constraint** − The CHECK constraint ensures that all the values in a column satisfies certain conditions.

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a single column it allows only certain values for this column.

- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that you can not have any person below 18 years:

- CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int CHECK (Age>=18));

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

- CREATE TABLE Persons (ID int NOT NULL,LastName varchar(255) NOT NULL, FirstName varchar(255), Age int,City varchar(255),CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes'));

SQL CHECK on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

- ALTER TABLE Persons ADD CHECK (Age>=18);

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

- ALTER TABLE Persons ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

- SQL Server / Oracle / MS Access: ALTER TABLE Persons DROP CONSTRAINT CHK_PersonAge;
- MySQL: ALTER TABLE Persons DROP CHECK CHK_PersonAge;

7. **INDEX** − Used to create and retrieve data from the database very quickly.

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

  CREATE INDEX Syntax

  Creates an index on a table. Duplicate values are allowed:

  CREATE INDEX index_name ON table_name (column1, column2, ...);

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index_name ON table_name (column1, column2, ...);

Note: The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

CREATE INDEX Example

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

- CREATE INDEX idx_lastname ON Persons (LastName);

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

- CREATE INDEX idx_pname ON Persons (LastName, FirstName); DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

- DROP INDEX index_name;

EX.NO: 6	VIEWS

AIM: To write an SQL query for views.

PROCEDURES:

Q1: Create a table employee with attributes empno, ename, job, mgr, hiredate, sal, commission, deptno with appropriate data type. Also create a table department with attributes deptno, dname, loc.

SQL> select * from employee1;

EMPNO	ENAME	MGRSSN	SALARY	DEPTNO	JOB
111	nirmala	3456	40000	2	lecturer
112	nandhu	5678	80000	1	surgeon
113	sathya	2345	90000	3	marketing
114	vino	6789	50000	2	doctor

SQL> select * from department;

DNO	DNAME	LOC
1	Medical	Madurai
2	english	Chennai
3	sales	Madurai

Q2: Create a view V1, which contain employee1 names and their manager names working in sales department

Query:

SQL> create view v1 as select ename,mgrssn from employee1 where deptno=(select deptno from department where dname='sales');

Output: View created.

SQL> select * from v1;

ENAME	MGRSSN
sathya	2345

Q3: Create a view called V2 which contains column such as empno, name, job from employee table. If V2 already exists your view command has to delete existing view and recreated it with the current query.

Query: SQL> create view v2 as select empno,ename,job from employee1;

Output: View created.

SQL> select * from v2;

EMPNO	ENAME	JOB
111	nirmala	lecturer
112	nandhu	surgeon
113	sathya	marketing
114	vino	doctor

Q4: Create a view V3 from employee table with check option on column sal>45000.

Query: SQL> create view v3 as select empno,ename,salary,job from employee1 where salary>50000;

Output: View created.

SQL> select * from v3;

EMPNO	ENAME	SALARY	JOB
111	nirmala	40000	Lecturer
112	nandhu	80000	Surgeon
113	sathya	90000	marketing
114	vino	50000	Doctor

Q5: Create a view called V4 for the table employee, which does not exists currently.

Query: SQL> create view v4 as select ename,job from employee1;

Output: View created.

SQL> select * from v4;

ENAME	JOB
nirmala	lecturer
nandhu	surgeon
sathya	marketing
vino	doctor

Q6: Remove all the views.

Query:

SQL> drop view v1; View dropped.

SQL> drop view v2; View dropped.

SQL> drop view v3; View dropped.

SQL> drop view v4; View dropped.

SQL> drop view v5; View dropped.

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

The WITH CHECK OPTION

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following code block has an example of creating same view CUSTOMERS_VIEW with

the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age FROM CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View

A view can be updated under certain conditions which are given below −

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain sub queries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

```
SQL > UPDATE CUSTOMERS_VIEW SET AGE = 35 WHERE name = 'Ramesh'
```

Inserting Rows into a View

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Deleting Rows into a View

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command. Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW WHERE age = 22;
```

EX.NO: 7	PL/SQL - FUNCTION AND PROCEDURES

AIM: To write a PL/SQL program by implementing procedures and functions.

Creating a Procedure

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

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;
```

Example:

```
CREATE OR REPLACE PROCEDURE
greetings AS
BEGIN
        Dbms_output.put_line(„HELLO  WORLD‟);
END;
/
```

Procedure created

To Execute the Procedure:

```
EXECUTE greetings;
```

OUTPUT:

HELLO WORLD

IN & OUT Mode Example 1

This program finds the minimum of two values. Here, the procedure takes two numbers using the IN mode and returns their minimum using the OUT parameters.

```
DECLARE
a number;
   b number;
    c number;
  PROCEDURE findMin(x IN number, y IN number, z OUT number)
  IS BEGIN
     IF x < y THEN
       z:=x;
     ELSE
       z:= y;
     END
     IF;
  END;
  BEGIN
     a:= 23;
     b:= 45;
     findMin(a, b, c);
     dbms_output.put_line(' Minimum of (23, 45) : ' ||c);
     END;
  /
```

OUTPUT:

Minimum of (23, 45) :23

PROCEDURE TO INSERT NUMBER

```
SQL> create table emp1(id number(3),First_name
varchar2(20)); Table created.

SQL> insert into emp1
values(101,'Nithya'); 1 row created.

SQL> insert into emp1
values(102,'Maya'); 1 row created.

SQL> select * from emp1;

        ID          FIRST_NAME
----------  ----------------------
       101          Nithya
       102          Maya
```

```
SQL>set server ouput on
SQL> create or replace procedure insert_num(p_numnumber)is
        begin
          insert into emp1(id,First_name) values(p_num,user);
          end
          insert_num;
          /

          Procedure created.

SQL> exec insert_num(3);
          PL/SQL procedure successfully completed.

SQL> select * from emp1;

          ID       FIRST_NAME
          ---------- ----------------------
          101      Nithya
          102      Maya
          103      SCOTT
```

The following example demonstrates Declaring, Defining, and Invoking a Simple PL/SQL Function that computes and returns the maximum of two values.

```
DECLARE
  a    number;
  b    number;
  c    number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
   z number;
BEGIN
  IF x > y THEN
    z:= x;
  ELSE
    Z:= y;
  END      IF;
  RETURN z;
END;
BEGIN
  a:= 23;
  b:= 45;
  c := findMax(a, b);
  dbms_output.put_line(' Maximum of (23,45): ' ||
c); END;
/
```

OUTPUT:

```
Maximum of (23,45):  45
```

FUNCTION TO FIND FACTORIAL

```
SQL>
  1 create or replace function
  fact(n number)
  2 return number is
  3 i number(10);
  4 f number:=1;
  5 begin
  6 for i in 1..N loop
  7 7 f:=f*i;
  8 end loop;
  9 return f;
  10 end;
  11 /
```

OUTPUT:

```
SQL> select fact(2) from dual;
```

PL/SQL PROGRAM FOR BONUS CALCULATION

SQL>set server output on;

```
SQL> declare
2 salary number;
3 bonus number;
4 begin
5 salary:=&sa;
6 if salary>5000 then
7  bonus:=salary*0. 5;
8 else
9 bonus:=0;
10 end if;
11 dbms_output.put_line(bonus);
12 end;
13 /
```

OUTPUT:

2500

EX.NO: 8	PL/SQL PROGRAM USING FOR LOOP

AIM: To write a PL/SQL program using For loop to insert ten rows into a database table

Program:

```
DECLARE
x NUMBER := 100;
BEGIN
FOR i IN 1..10 LOOP
IF MOD(i,2) = 0 THEN     -- i is even
INSERT INTO temp VALUES (i, x, 'i is even');
ELSE
INSERT INTO temp VALUES (i, x, 'i is odd');
END IF;
x := x + 100;
END LOOP;
COMMIT;
END;
```

Output Table

SQL> SELECT * FROM temp ORDER BY col1;

COL1	COL2	MESSAGE
1	100	i is odd
2	200	i is even
3	300	i is odd
4	400	i is even
5	500	i is odd
6	600	i is even
7	700	i is odd
8	800	i is even
9	900	i is odd
10	1000	i is even

10 records inserted.

EX.NO: 9	RELATIONAL ALGEBRA QUERIES

AIM: To write a SQL query to perform all relational algebra operations from set theory.

PROCEDURE:

„Relational Algebra Operations From Set Theory

- UNION ( )
- INTERSECTION (∩),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (X)

1.UNION OPERATION ()

-

UNION is symbolized by symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A B

For a union operation to be valid, the following conditions must hold -

R and S must be the same number of attributes.

Duplicate tuples should be automatically removed.

Example

Consider the following tables.

Table A

column 1	column 2
1	1
1	2

Table B

column 1	column 2
1	1
1	3

A **B gives : Table A B**

column 1	column 2
1	1
1	2
1	3

Set Difference (-)

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

The attribute name of A has to match with the attribute name in B.

The two-operand relations A and B should be either compatible or Union compatible.

It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example

A-B : Table A – B

column 1	column 2
1	2

Intersection

An intersection is defined by the symbol ∩

Example:

A ∩ B : Table A ∩ B

column 1	column 2
1	1

Cartesian Product(X):

Cartesian Product in DBMS is an operation used to merge columns from two relations. Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations. It is also called Cross Product or Cross Join.

Example – Cartesian product

σ column 2 = '1' (A X B)

Output – The above example shows all rows from relation A and B whose column 2 has value 1 σ column 2 = '1' (A X B)

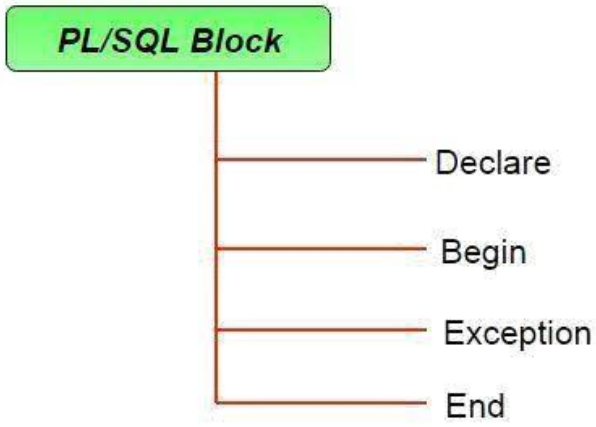
column 1	column 2
1	1
1	1

EX.NO: 10	PL/SQL - ONE BLOCK USING SINGLE COMMAND

AIM: To write a PL/SQL to execute a number of queries in one block using single command

Structure of PL/SQL Block:

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.



Typically, each block performs a logical action in the program. A block has the following structure:

```
DECLARE
  declaration statements;

BEGIN
  executable statements

EXCEPTIONS
  exception handling statements

END;
```

- Declare section starts with **DECLARE** keyword in which variables, constants, records as cursors can be declared which stores data temporarily. It basically consists definition of PL/SQL identifiers. This part of the code is optional.
- Execution section starts with **BEGIN** and ends with **END** keyword.This is a mandatory section and here the program logic is written to perform any task like loops and conditional statements. It supports all DML commands, DDL commands and SQL*PLUS built-in functions as well.

- Exception section starts with **EXCEPTION** keyword.This section is optional which contains statements that are executed when a run-time error occurs. Any exceptions can be handled in this section.

Explanation:

- **SET SERVEROUTPUT ON**: It is used to display the buffer used by the dbms_output.
- **var1 INTEGER :** It is the declaration of variable, named ***var1*** which is of integer type. There are many other data types that can be used like float, int, real, smallint, long etc. It also supports variables used in SQL as well like NUMBER(prec, scale), varchar, varchar2 etc.
- **PL/SQL procedure successfully completed.:** It is displayed when the code is compiled and executed successfully.
- **Slash (/) after END;:** The slash (/) tells the SQL*Plus to execute the block.

PL/SQL code to print sum of two numbers taken from the user.

```
SQL> SET SERVEROUTPUT ON;

SQL> DECLARE

   -- taking input for variable a
   a  integer := &a ;
   -- taking input for  variable  b
   b integer := &b ;
   C  integer ;

 BEGIN
   c: = a + b ;
   dbms_output.put_line ('Sum of '||a||' and '||b||' is = '||c);

 END;
 /
```

OUTPUT:

```
1
2
Sum of a and b is = 3
```