

SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur – 603 203

DEPARTMENT OF CYBER SECURITY

Lab Manual

CS3466 - DATABASE MANAGEMENT SYSTEMS LABORATORY

Regulation 2023



II Year (IV Semester)

Academic Year: 2024 – 2025 (EVEN SEMESTER)

Prepared by

Dr. G. Kumaresan, Associate Professor / CSE

Dr. M. Mayuranathan, Associate Professor / CSE

Ms. S. Anslam Sibi / A.P - Sr.G / CSE

PROGRAMME OUTCOMES

- 1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- 2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using the first principles of mathematics, natural sciences, and engineering sciences.
- 3. Design/development of solutions:** Design solutions for the 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.
- 4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including the design of experiments, analysis and interpretation of data, and the synthesis of the information to provide valid conclusions.
- 5. Modern 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.
- 6. 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.
- 7. Environment 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.
- 8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- 9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- 10. Communication:** 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.
- 11. 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.

12. 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.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: Exhibit proficiency in planning, implementing and evaluating team oriented-software programming solutions to specific business problems and society needs.

PSO2: Demonstrate professional skills in applying programming skills, competency and decision making capability through hands-on experiences.

PSO3: Apply logical thinking in analyzing complex real world problems, and use professional and ethical behaviors to provide proper solutions to those problems.

PSO4: Demonstrate the ability to work effectively as part of a team in applying technology to Business and personal situations.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: To mould students to exhibit top performance in higher education and research and to become a State-of –the- art technocrat.

PEO2: To impart the necessary background in Computer Science and Engineering by providing solid foundation in Mathematical, Science and Engineering fundamentals.

PEO3: To equip the students with the breadth of Computer Science and Engineering innovate novel solutions for the benefit of common man.

CO – PO and PSO MAPPING:

CO's	PO's												PSO's				
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	
1	3	3	3	3	-	-	-	-	3	1	3	2	2	3	2	-	
2	2	2	3	2	2	-	-	-	1	2	3	3	2	1	2	-	
3	3	3	2	1	1	-	-	-	1	1	1	3	2	3	3	-	
4	1	3	3	3	1	-	-	-	1	1	3	2	3	1	3	-	
5	3	2	1	1	1	-	-	-	2	2	3	1	3	1	2	-	
AVG	2	3	2	2	1	-	-	-	2	1	3	2	2	2	2	-	

CS3466 DATABASE MANAGEMENT SYSTEMS LABORATORY L TP C
0035

COURSE OBJECTIVES:

- To learn and implement important commands in SQL.
- To learn the usage of nested and joint queries.
- To understand functions, procedures and procedural extensions of databases.
- To understand design and implementation of typical database applications.
- To be familiar with the use of a front end tool for GUI based application development.

LIST OF EXPERIMENTS

1. Create a database table, add constraints (primary key, unique, check, Not null), insert rows, update and delete rows using SQL DDL and DML commands.
 2. Create a set of tables, add foreign key constraints and incorporate referential integrity.
 3. Query the database tables using different 'where' clause conditions and also implement aggregate functions.
 4. Query the database tables and explore sub queries and simple join operations.
 5. Query the database tables and explore natural, equi and outer joins.
 6. Write user defined functions and stored procedures in SQL.
 7. Execute complex transactions and realize DCL and TCL commands.
 8. Write SQL Triggers for insert, delete, and update operations in a database table.
 9. Create View and index for database tables with a large number of records.
 10. Case Study using any of the real life database applications from the following list
 - a) Inventory Management for a EMart Grocery Shop
 - b) Society Financial Management
 - c) Cop Friendly App – Eseva
 - d) Property Management – eMall
 - e) Star Small and Medium Banking and Finance
- Build Entity Model diagram. The diagram should align with the business and functional goals stated in the application.
 - Apply Normalization rules in designing the tables in scope.
 - Prepared applicable views, triggers (for auditing purposes), and functions for enabling enterprise grade features.

List of Equipment :(30 Students per Batch)
MYSQL / SQL : 30 Users

TOTAL: 45 PERIODS

INDEX

Ex. NO	LIST OF EXPERIMENTS:	PAGE NO
	Instructions for oracle Commands, syntaxes for viva voce	3
1	Creation of a database and writing SQL queries to Retrieve information from the database.	9
2	Performing insertion, deletion, modifying, altering, Updating and viewing records based on conditions.	11
3	Creating an Employee database to set various Constraints and Creation of Views Indexes, Save point.	19
4	Joins and Nested Queries.	23
5	Study of PL/SQL block.	29
6	Write a PL/SQL block to satisfy some conditions by accepting input from the user.	32
7	Write a PL/SQL block that handles all types of Exceptions.	33
8	Creation of Procedures.	37
9	Creation of database triggers and functions	42
10	Creation of Database in MS Access.	44
11	Database connectivity using Front End Tools (Application Development using Oracle/ mysql)	49
	Mini project	

INSTRUCTIONS FOR ORACLE COMMANDS, SYNTAXES FOR VIVA VOCE

SQL consists of a small number of high-level commands that let you query a database, and even build new databases.

- Tables are the basic building blocks of a database.
- Columns define the categories of information in the table
- Rows represent individual records in the table.

- **SQL is provided in two modes.**

- Interactive SQL

- This mode is used to operate directly on a database that is the response to any SQL command can be seen almost immediately on the same terminal.

- Embedded SQL

- Embedded SQL consists of SQL commands used within programs written in some other language like COBOL, PASCAL or C

SQL features

1. It is a unified language.
2. It is common language for relational database
3. It is a non-procedural language.

SQL Language commands

1. Data Definition Language [DDL] – Create, Alter, Drop
2. Data Manipulation Language [DML]– Insert, Update, Delete
3. Transaction Control Language [TCL] – Commit, RollBack, Savepoint

ORACLE DATA TYPES

1. Char(n)
2. varchar2(n)
3. Number(p , s)
4. Date
5. Raw(n)

1) **Char(n)** – It is used for fixed length character data of length 'n' at maximum bytes of 255

-n is used for number of character(s)

2) **Varchar2(n)** - It is used for variable length character data . A max. n (column 2000 bytes in length) must be specified.

3) **Number(P,S)** – It is used for variable numeric data with Precision P & Scales S.

Eg. Salary Number(10,3)

Here, the number values up to 10 digits wide, three of the digits following the decimal point.

4) **Date** - It is used for fixed length date & time data - 1-JAN-4712 BC to 31-DEC-4712 AD

6. **Raw(n)** – Binary data of max. n (max. 255 bytes)

7. **Long** - It is used for variable length character data at a maximum of $2^{31} - 1$ bytes

Rules for naming a TABLE: All the rules for naming a variable in a high level language will apply to table's name also.

- 1) Must begin with an alphabet (ie) A-Z or a-z

- 2) May contain letters, numerals and the special characters, _(underscore). It is advisable to avoid the usage of \$ and # symbols).
- 3) Not case sensitive. The length of the table name may extend up to 30 characters in length. Eg. 1) Dept 2)DEPT 3)dept
- 4) The table name should be unique
- 5) Should not be an ORACLE reserved word
- 6) Blank spaces, commas are not allowed.
- 7) No two columns in the same table have the same column name.

DATA DEFINITION LANGUAGE (DDL)

DDL consists of three SQL commands.

1. CREATE
2. ALTER
3. DROP

Data Manipulation Language (DML)

The DML consists of four SQL commands.

- 1) INSERT
- 2) SELECT
- 3) UPDATE
- 4) DELETE

TRANSACTION CONTROL LANGUAGE(TCL)

- A transaction is not made permanent in ORACLE database unless it is committed or until it executes an ALTER, AUDIT, CREATE, DISCONNECT, DROP, NEXT, GRANT, NO AUDIT, QUIT OR REVOKE.

TCL commands are

1. COMMIT
2. ROLLBACK
3. SAVEPOINT

COMMIT:

- It is not necessary to have any privileges to commit current transaction.
- The COMMIT (save with recent changes) command forces SQL to commit pending table changes to the database.
- It is good practice to commit changes to the database as soon as you finish a work and at frequent intervals.

Syntax:

SQL>COMMIT WORK;(Press enter key)
SQL>COMMIT; (Press enter key)

ROLLBACK

- To undo work done in the current transactions
- Rolling back means undoing any changes to data that have been performed by SQL statements within an uncommitted transaction.
- To roll back with savepoint_id
 - Rollback the current transaction to the specified savepoint.
 - If omitted, the ROLLBACK statement roll back the entire transaction.
 - Savepoint_id is an valid character string.

Syntax:

SQL>ROLLBACK WORK; (Press enter key)

SQL> ROLLBACK; (Press enter key)

Work is optional

SAVEPOINT:

To identify a point in a transaction to which you can later rollback.

Savepoints are often used to divide a long transaction into smaller parts.

Syntax:

SQL>SAVEPOINT <savepoint_id>; (Press enter key)

Example: SQL>SAVEPOINT R;(Press enter key)

Output: Savepoint created.

PRIVILEGE COMMANDS (Data Control Commands)

Privilege commands are

- 1) Grant
- 2) Revoke

Some of the privileges & objects are

Privilege	Object
SELECT	Data in a table in or view
INSERT	Rows into a table or view
UPDATE	Values in a table or view
DELETE	Rows from a table or view
ALTER	Column definitions in a table
INDEX	A column in a table or view

Grant: If one user wants to share another user's table the privilege should be given first

Syntax:

SQL>GRANT <privileges> ON <table name> TO <user name> ; (Press enter key)

Granting Privileges: To grant a user the privilege to select from our table name.

Syntax:

SQL> GRANT SELECT ON DEPT TO GANESH;(Press enter key)

Output Result: Grant succeeded

Note: Here, GANESH is another user. The above message grant succeeded tells you that the privilege has been granted

Passing privileges

When you grant an access privilege, the user who receives the grant normally does not receive authority to pass the privilege onto others.

To give user a authority to pass privileges use the clause with GRANT option.

SQL>GRANT SELECT ON DEPT TO GANESH WITH GRANT OPTION;(Press enter

key)

Output: Grant Succeeded.

REVOKE: To withdraw a privilege you have granted, use the revoke command.

Syntax:

SQL> REVOKE <privileges> ON <table or view> FROM <users>; (Press enter key)

- When you user revoke, the privileges you specify are revoked from the users you name and from any other users to whom they have granted those privileges.

Example

SQL>REVOKE SELECT ON DEPT FROM GANESH; (Press enter key)

Output: Revoke Succeeded.

Cursors

A *cursor* is a variable that runs through the tuples of some relation. This relation can be a stored table, or it can be the answer to some query. By fetching into the cursor each tuple of the relation, we can write a program to read and process the value of each such tuple. If the relation is stored, we can also update or delete the tuple at the current cursor position.

syntax

CURSOR cursor_name IS select_statement;

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:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
```

```
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
```

```
{IS | AS}
```

```
BEGIN
```

```
< procedure_body >
```

```
END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows modifying an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Function

A PL/SQL function is same as a procedure except that it returns a value. Therefore, all the discussions of the previous chapter are true for functions too.

Creating a Function

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows:

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
< function_body >
END [function_name];
Where,
```

- *function-name* specifies the name of the function.
- [OR REPLACE] option allows modifying an existing function.

- The optional parameter list contains name, mode and types of the parameters. IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- The function must contain a **return** statement.
- *RETURN* clause specifies that data type you are going to return from the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

Triggers

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes:

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
Declaration-statements
BEGIN
Executable-statements
EXCEPTION
Exception-handling-statements
END;
Where,
```

- CREATE [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF}: This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Ex No. 1

SQL – Structured Query Language

Date:

Aim: To create database tables and views using Oracle.

Procedure:

1) CREATE: This command helps to create a table

Syntax: SQL> CREATE TABLE <table-name> (Column-element1 datatype, column-element2 datatype....)

Eg. SQL> CREATE TABLE DEPT(deptno number(2), deptname varchar2(5), loc char2(8)); (Press enter key)

If you want to see the structure of the table

SQL> DESCRIBE <table-name> (Press enter key)

SQL>DESC <table-name> (Press enter key)

2) ALTER – used to add a new column or modify the width of an existing column in a table

Syntax: 1) With MODIFY command (MODIFY –oracle reserved word)

SQL> ALTER <table-name> MODIFY (column-definitions) (Press Enter Key)

Example

SQL>ALTER TABLE DEPT MODIFY(DEPTNAME VARCHAR2(20)); (Press enter key)

Syntax: 2) With ADD command (ADD – Oracle Reserved word) is used to add column/s) in a table.

SQL>ALTER TABLE <table-name> ADD(column-definitions); (Press enter key)

Example;

SQL>ALTER TABLE EMP ADD(ADDRESS CHAR(30)); (Press enter key)

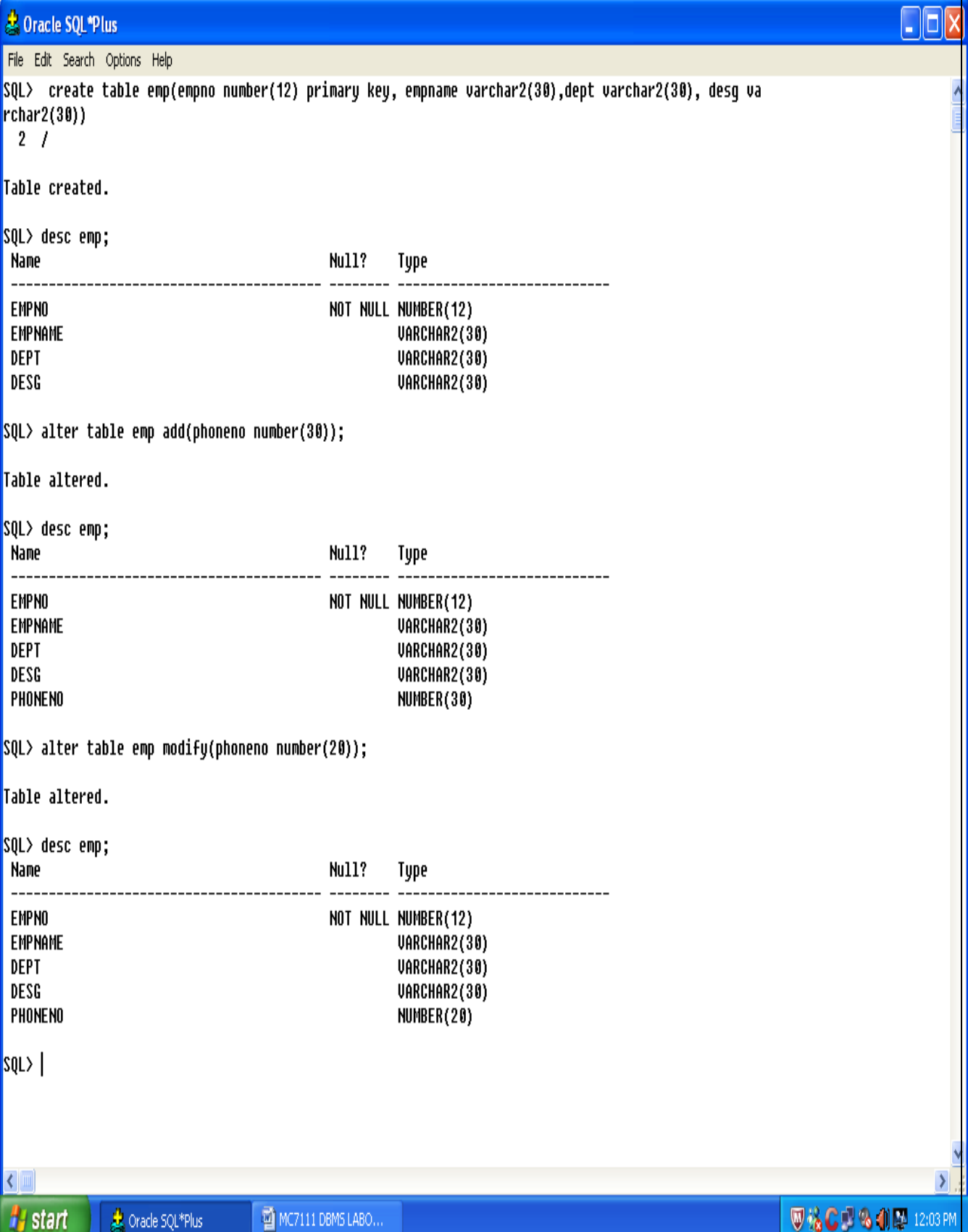
-Column to be modified must be empty to decrease precision or scale

3) DROP: To delete the table values with structure

Syntax: SQL> DROP TABLE <tablename>; (press enter key)

Example : SQL>DROP TABLE DEPT;

OUTPUT



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> create table emp(empno number(12) primary key, empname varchar2(30),dept varchar2(30), desg va
rchar2(30))
  2 /

Table created.

SQL> desc emp;
Name                                Null?    Type
-----
EMPNO                                NOT NULL NUMBER(12)
EMPNAME                               VARCHAR2(30)
DEPT                                  VARCHAR2(30)
DESG                                  VARCHAR2(30)

SQL> alter table emp add(phoneno number(30));

Table altered.

SQL> desc emp;
Name                                Null?    Type
-----
EMPNO                                NOT NULL NUMBER(12)
EMPNAME                               VARCHAR2(30)
DEPT                                  VARCHAR2(30)
DESG                                  VARCHAR2(30)
PHONENO                               NUMBER(30)

SQL> alter table emp modify(phoneno number(20));

Table altered.

SQL> desc emp;
Name                                Null?    Type
-----
EMPNO                                NOT NULL NUMBER(12)
EMPNAME                               VARCHAR2(30)
DEPT                                  VARCHAR2(30)
DESG                                  VARCHAR2(30)
PHONENO                               NUMBER(20)

SQL> |
```

The screenshot shows the Oracle SQL*Plus interface with a blue title bar and a menu bar (File, Edit, Search, Options, Help). The main window contains a SQL prompt and several commands. The first command creates a table 'emp' with columns 'empno' (primary key, NUMBER(12)), 'empname' (VARCHAR2(30)), 'dept' (VARCHAR2(30)), and 'desg' (VARCHAR2(30)). The second command describes the table, showing the column names, nullability, and data types. The third command adds a 'phoneno' column (NUMBER(30)) to the table. The fourth command describes the table again, showing the new column. The fifth command modifies the 'phoneno' column to NUMBER(20). The sixth command describes the table once more, showing the final state. The prompt 'SQL> |' is visible at the bottom of the window. The Windows taskbar at the bottom shows the Start button, the Oracle SQL*Plus application, and a taskbar with the text 'MC7111 DBMS LABO...'. The system tray on the right shows various icons and the time '12:03 PM'.

Result:

Thus the above experiment was successfully completed.

Ex No. 2

Data Manipulation Language (DML)

Date:

Aim: To perform insert, update, delete and query operations in database tables.

Procedure:

1) INSERT

SQL> INSERT INTO table-name VALUES (a list of data values); (Press enter key)

Example : Method-1

SQL>INSERT INTO EMP VALUES(396,'RAMA',300,5000,200,'6-JUN-59'); (press enter key)

Note: Date and character data-type values should be enclosed in quotes

Example: Method-2

If we want to insert only empno and age the command would be

SQL>INSERT INTO EMP(ENAME, AGE) VALUES(396,38); (Press enter key)

Example : Method-3

We can insert into one table by copying rows another table, by using “select” statement.

SQL> INSERT INTO EMP(ENAME,JOB,SAL,COMM) SELECT ENAE, JOB,SAL, COMM FROM EMP WHERE DESIGN =“SALESMAN”; (Press enter key)

Method-4

SQL>INSERT INTO <table-name> values('&empno','&empname', -----); (press enter key)

QUERY –A query is a request for information.

2) SELECT

Syntax

SQL>SELECT column-name1, column-name2 _____ FROM table-name1, table-name2 _ _ _ ;(Press enter key)

Example

SQL> SELECT EMPNAME, AGE FROM EMP; (Press enter key)

<u>OUTPUT</u>	<u>EMPNAME</u>	<u>AGE</u>
	RAMESH	24
	SURESH	20
	SATISH	30

SQL>SELECT * FROM EMP; (Press enter key)

->displays all rows and columns in the table ‘emp”

OUTPUT

<u>EMPNO</u>	<u>EMPNAME</u>	<u>AGE</u>	<u>SALARY</u>
1001	RAMESH	24	10000
1002	SURESH	20	8000

⋮

CHANGING COLUMN ORDER: The order of column name in a select command determines the order in which the columns are displayed.

Example 1: SQL> SELECT EMPNO,AGE FROM EMP;

Example 2: SQL> SELECT AGE,EMPNO FROM EMP;

SQL> SELECT JOB FROM EMP; (Press Enter key)

Output

JOB
ASSISTANT
SUPDT
ASSISTANT
HELPER
MECHANIC
SUPDT
CLERK

To eliminate duplicate rows in the result, include the distinct clause in the 'select' command

SQL>SELECT DISTINCT JOB FROM EMP;(Press enter key)

JOB
ASSISTANT
SUPDT
HELPER
MECHANIC
CLERK

SELECT command with WHERE clause

Syntax:

SELECT columns FROM table-name WHERE logical conditions to be met; (Press Enter key)

Example:

SQL>SELECT ENAME FROM EMP WHERE DEPT='CSG'; (Press enter key)

UPDATE

- To change the value entered in the given table
- SET clause and optional WHERE clause.
- To update one or many rows in a table

Method -1 WHERE clause

Syntax:

SQL>UPDATE tablename SET field = value, field= value, WHERE logical expressions; (Press enter key)

Example

SQL>UPDATE EMP SET AGE=45 WHERE ENAME="RAJA"; (Press enter key)

Method-2 Arithmetic Operations

Example

SQL>UPDATE EMP SET SALARY = SALARY *0.25 + SALARY; (Press enter key)

Method-3 UPDATE with another table

Example

SQL>UPDATE EMP SET SALARY = SALARY* 1.15 WHERE ENAME IN (SELECT ENAME FROM BONUS); (Press enter key)

DELETE

- Used to delete rows from a table
- Contains FROM clause followed by optional WHERE clause
- One or more rows can be deleted at a time. Deletion of single column element is not possible

Method-1: To delete a particular column with WHERE clause

Syntax:

SQL>DELETE FROM table-name WHERE <logical conditions>; (Press Enter Key)

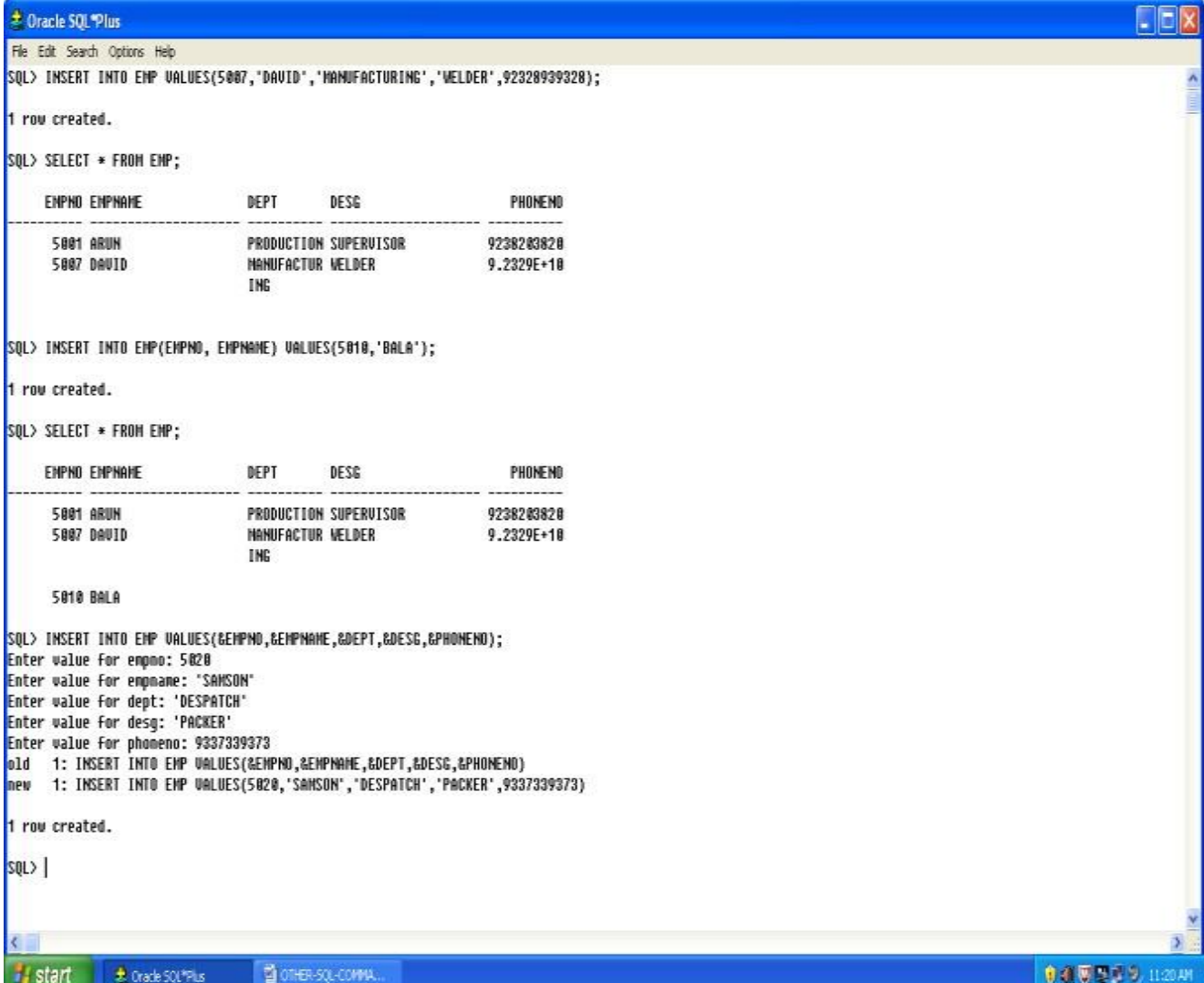
Example:

SQL>DELETE FROM EMP WHERE CODE=5; (Press enter key)

Method-2: To delete all rows in a table.

Example: SQL> DELETE FROM EMP; (Press Enter Key)

OUTPUT:



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> INSERT INTO EMP VALUES(5007,'DAVID','MANUFACTURING','WELDER',92328939328);
1 row created.
SQL> SELECT * FROM EMP;
  EMPNO EMPNAME          DEPT          DESG          PHONENO
-----
  5001 ARUN              PRODUCTION SUPERVISOR          9238203820
  5007 DAVID              MANUFACTUR WELDER              9.2329E+10
          INC

SQL> INSERT INTO EMP(EMPNO, EMPNAME) VALUES(5010,'BALA');
1 row created.
SQL> SELECT * FROM EMP;
  EMPNO EMPNAME          DEPT          DESG          PHONENO
-----
  5001 ARUN              PRODUCTION SUPERVISOR          9238203820
  5007 DAVID              MANUFACTUR WELDER              9.2329E+10
          INC

  5010 BALA

SQL> INSERT INTO EMP VALUES(&EMPNO,&EMPNAME,&DEPT,&DESG,&PHONENO);
Enter value for empno: 5020
Enter value for empname: 'SAHSON'
Enter value for dept: 'DESPATCH'
Enter value for desg: 'PACKER'
Enter value for phoneno: 9337339373
old 1: INSERT INTO EMP VALUES(&EMPNO,&EMPNAME,&DEPT,&DESG,&PHONENO)
new 1: INSERT INTO EMP VALUES(5020,'SAHSON','DESPATCH','PACKER',9337339373)
1 row created.
SQL> |
```

```
SQL> select * from test;
```

REGNO	NAME	MARK1	MARK2
5001	ARUN	60	40
5040	SANKAR	65	40
5005	DHILIP	40	40

```
SQL> INSERT INTO EMP(EMPNO,EMPNAME) SELECT REGNO, NAME FROM TEST WHERE REGNO=5040;
```

1 row created.

```
SQL> SELECT * FROM EMP;
```

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTURING	WELDER	9.2329E+10
5010	BALA	DESPATCH	PACKER	9337339373
5020	SANSON			
5040	SANKAR			

```
SQL> |
```



SQL> SELECT * FROM EMP;

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTUR	WELDER	9.2329E+10
		ING		
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR			

SQL> SELECT EMPNAME,PHONENO FROM EMP;

EMPNAME	PHONENO
ARUN	9238203820
DAVID	9.2329E+10
BALA	
SAMSON	9337339373
SANKAR	

SQL> SELECT DEPT,EMPNAME,PHONENO FROM EMP;

DEPT	EMPNAME	PHONENO
PRODUCTION	ARUN	9238203820
MANUFACTUR	DAVID	9.2329E+10
ING		
	BALA	
DESPATCH	SAMSON	9337339373
	SANKAR	

SQL> UPDATE EMP SET DEPT='DESPATCH' WHERE EMPNAME='SANKAR';

1 row updated.

```
SQL> UPDATE EMP SET DEPT='DESPATCH' WHERE EMPNAME='SANKAR';
```

```
1 row updated.
```

```
SQL> SELECT * FROM EMP;
```

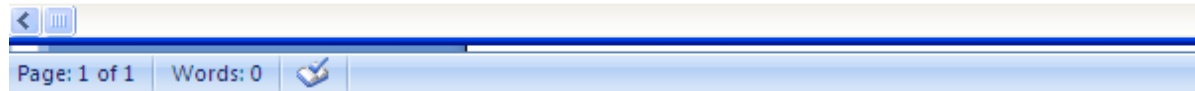
EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTUR	WELDER	9.2329E+10
		ING		
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH		

```
SQL> SELECT DISTINCT DEPT FROM EMP;
```

```
DEPT
-----
DESPATCH
MANUFACTUR
ING

PRODUCTION
```

```
SQL>
```



```
SQL> SELECT * FROM EMP;
```

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTUR	WELDER	9.2329E+10
		ING		
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH		

```
SQL> SELECT * FROM TEST;
```

REGNO	NAME	MARK1	MARK2
5001	ARUN	60	40
5040	SANKAR	65	40
5005	DHILIP	40	40

```
SQL> UPDATE EMP SET DESG='SUPERVISOR' WHERE EMPNAME IN (SELECT NAME FROM TEST WHERE MARK1=65);
```

```
1 row updated.
```

```
SQL> SELECT * FROM EMP
```

```
2 /
```

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTUR	WELDER	9.2329E+10
		ING		
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH	SUPERVISOR	

```
SQL> |
```

```
SQL> SELECT * FROM TEST;
```

REGNO	NAME	MARK1	MARK2
5001	ARUN	60	40
5040	SANKAR	65	40
5005	DHILIP	40	40

```
SQL> DELETE FROM TEST WHERE NAME= 'DHILIP';
```

```
1 row deleted.
```

```
SQL> CREATE TABLE TEST1 AS SELECT * FROM TEST;
```

```
Table created.
```

```
SQL> SELECT * FROM TEST;
```

REGNO	NAME	MARK1	MARK2
5001	ARUN	60	40
5040	SANKAR	65	40

```
SQL> SELECT * FROM TEST1;
```

REGNO	NAME	MARK1	MARK2
5001	ARUN	60	40
5040	SANKAR	65	40

```
SQL> DELETE FROM TEST1;
```

```
2 rows deleted.
```

```
SQL> SELECT * FROM TEST1;
```

```
no rows selected
```

```
SQL>
```

Result:

Thus the above experiment was successfully completed.

Ex No. 3 Data Control Language (DCL) and Transaction Control Language (TCL)

Date:

Aim: To demonstrate DCL and TCL commands

Procedure:

TCL commands

- 1) Create a table
- 2) Insert records into the table
- 3) Using SELECT command check the inserted records
- 4) Type the command rollback and press enter key
- 5) Again check the records in the table using SELECT command. Inserted records will not be there in the table.
- 6) Now one record into the table and then place a savepoint using SAVEPOINT command.
- 7) Now insert two records and then execute a rollback.
- 8) Check the records in the table using SELECT command. You will have one record in the table. Last two records inserted will not be there.
- 9) Now one record into the table and then execute COMMIT command.
- 10) Now if you execute roll back no records will be deleted. Because COMMIT command will save all the previous transactions.

DCL commands

- 1) Let A be a super user and B be a ordinary user.
- 2) 'A' Log in as super user
- 3) Using Grand command grant SELECT privilege to user B
- 4) 'B' Log in as ordinary user and can use Select command to display the records of super user A
- 5) If B tries to use update command (or any command other than SELECT command) then error message indicating insufficient privileges will be displayed.
- 6) Superuser A can execute any command on the table of ordinary user.
- 7) Superuser can revoke the privileges granted to Ordinary user using REVOKE command.

OUTPUT

SQL> SELECT * FROM EMP;

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTURING	WELDER	9.2329E+10
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH	SUPERVISOR	

SQL> INSERT INTO EMP VALUES(5050,'JOKER','BILLING','CLERK',9234234838);

1 row created.

SQL> SELECT * FROM EMP;

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTURING	WELDER	9.2329E+10
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH	SUPERVISOR	
5050	JOKER	BILLING	CLERK	9234234838

6 rows selected.

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT * FROM EMP;

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTURING	WELDER	9.2329E+10
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH	SUPERVISOR	

SQL>

Rollback complete.

SQL> SELECT * FROM EMP;

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTURING	WELDER	9.2329E+10
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH	SUPERVISOR	

SQL> INSERT INTO EMP VALUES(5060,'RAM','QUALITY CONTROL','ENGINEER',9483837483);

1 row created.

SQL> SAVEPOINT S1;

Savepoint created.

SQL> SELECT * FROM EMP;

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTURING	WELDER	9.2329E+10
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH	SUPERVISOR	
5060	RAM	QUALITY CONTROL	ENGINEER	9483837483

6 rows selected.

SQL> DELETE FROM EMP WHERE EMPNO=5060;

1 row deleted.

SQL> SAVEPOINT S2;

Savepoint created.

SQL> DELETE FROM EMP WHERE EMPNO=5060;

1 row deleted.

SQL> ROLLBACK TO S2;

Rollback complete.

SQL> SELECT * FROM EMP;

EMPNO	EMPNAME	DEPT	DESG	PHONENO
5001	ARUN	PRODUCTION	SUPERVISOR	9238203820
5007	DAVID	MANUFACTURING	WELDER	9.2329E+10
5010	BALA			
5020	SAMSON	DESPATCH	PACKER	9337339373
5040	SANKAR	DESPATCH	SUPERVISOR	
5060	RAM	QUALITY CONTROL	ENGINEER	9483837483

6 rows selected.

SQL>

```
SQL> SHOW USER
USER is "PRINCE"
SQL> CONNECT LEO/LEO@DBSERVER
Connected.
SQL> SELECT * FROM PRINCE.STUDENT;
SELECT * FROM PRINCE.STUDENT
*
```

ERROR at line 1:
ORA-00942: table or view does not exist

```
SQL> CONNECT PRINCE/JAMES@DBSERVER;
Connected.
SQL> GRANT SELECT ON STUDENT TO LEO;

Grant succeeded.
```

```
SQL> CONNECT LEO/LEO@DBSERVER;
Connected.
SQL> SELECT * FROM PRINCE.STUDENT;
```

REGNO	NAME	DEPT			M1	M2
M3	M4	M5	TOTAL	AVERAGE RESU G	RANK	
4.2209E+10	Annie		MCA		65	76
80	68	67	356	71.2	2	
4.2209E+10	Prince		MCA			
4.2209E+10	Divya S		MCA		61	77
66	79	68	351	70.2	3	

REGNO	NAME	DEPT			M1	M2
M3	M4	M5	TOTAL	AVERAGE RESU G	RANK	
4.2209E+10	Elayaraja T		MCA		51	63
60	71	53	298	59.6	7	
4.2209E+10	Ezhilarasan D		MCA		61	72
65	74	59	331	66.2	5	
4.2209E+10	Shanmuga Priya		IT			

12 rows selected.

```
SQL> UPDATE PRINCE.STUDENT SET NAME='Divya S';
UPDATE PRINCE.STUDENT SET NAME='Divya S'
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

```
SQL> |
```

Result:

Thus the above experiment was successfully completed.

Ex. No. 4 JOINS AND NESTED QUERIES

Date:

Aim: To demonstrate Joins and Nested Queries

Procedure

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use STUDENT, COURSE, STUDENT_COURSE tables for understanding nested queries.

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S_AGE
S1	RAM	DELHI	9455123451	18
S2	RAMESH	GURGAON	9652431543	18
S3	SUJIT	ROHTAK	9156253131	20
S4	SURESH	DELHI	9156768971	18

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID
S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

There are mainly two types of nested queries:

Independent Nested Queries: In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

IN: If we want to find out S_ID who are enrolled in C_NAME 'DSA' or 'DBMS', we can write it with the help of independent nested query and IN operator. From COURSE table, we can find out C_ID for C_NAME 'DSA' or 'DBMS' and we can use these C_IDs for finding S_IDs from STUDENT_COURSE TABLE.

STEP 1: Finding C_ID for C_NAME = 'DSA' or 'DBMS'

```
Select C_ID from COURSE where C_NAME = 'DSA' or C_NAME = 'DBMS'
```

STEP 2: Using C_ID of step 1 for finding S_ID

```
Select S_ID from STUDENT_COURSE where C_ID IN
```

```
(SELECT C_ID from COURSE where C_NAME = 'DSA' or C_NAME='DBMS');
```

The inner query will return a set with members C1 and C3 and outer query will return those S_IDs for which C_ID is equal to any member of set (C1 and C3 in this case). So, it will return S1, S2 and S4.

Note: If we want to find out names of STUDENTs who have either enrolled in 'DSA' or 'DBMS', it can be done as:

```
Select S_NAME from STUDENT where S_ID IN
```

```
(Select S_ID from STUDENT_COURSE where C_ID IN
```

```
(SELECT C_ID from COURSE where C_NAME='DSA' or C_NAME='DBMS'));
```

NOT IN: If we want to find out S_IDs of STUDENTs who have neither enrolled in 'DSA' nor in 'DBMS', it can be done as:

```
Select S_ID from STUDENT where S_ID NOT IN
```

```
(Select S_ID from STUDENT_COURSE where C_ID IN
```

```
(SELECT C_ID from COURSE where C_NAME='DSA' or C_NAME='DBMS'));
```

The innermost query will return a set with members C1 and C3. Second inner query will return those S_IDs for which C_ID is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those S_IDs where S_ID is not a member of set (S1, S2 and S4). So it will return S3.

Co-related Nested Queries: In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. e.g.; If we want to find out S_NAME of STUDENTs who are enrolled in C_ID 'C1', it can be done with the help of co-related nested query as:

Select S_NAME from STUDENT S where EXISTS

(select * from STUDENT_COURSE SC where S.S_ID=SC.S_ID and SC.C_ID='C1');

For each row of STUDENT S, it will find the rows from STUDENT_COURSE where S.S_ID = SC.S_ID and SC.C_ID='C1'. If for a S_ID from STUDENT S, atleast a row exists in STUDENT_COURSE SC with C_ID='C1', then inner query will return true and corresponding S_ID will be returned as output.

JOIN OPERATIONS

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

INNER JOIN

LEFT JOIN

RIGHT JOIN

FULL JOIN

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

INNER JOIN: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1
```

```
INNER JOIN table2
```

```
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.

Example Queries(INNER JOIN)

This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
```

```
INNER JOIN StudentCourse
```

```
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

LEFT JOIN: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.

Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL

RIGHT JOIN: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1  
RIGHT JOIN table2  
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Result : Thus the above experiment was successfully completed.

Ex. No. 5**High level language extensions – PL/SQL****Date:****Aim:** To write simple program using PL/SQL**Procedure:**

```
CREATE TABLE T1 (
```

```
    e INTEGER,
```

```
    f INTEGER
```

```
);
```

```
DELETE FROM T1;
```

```
INSERT INTO T1 VALUES (1, 3);
```

```
INSERT INTO T1 VALUES (2, 4);
```

```
/* Above is plain SQL; below is the PL/SQL program. */
```

```
DECLARE
```

```
    a NUMBER;
```

```
    b NUMBER;
```

```
BEGIN
```

```
    SELECT e,f INTO a,b FROM T1 WHERE e>1;
```

```
    INSERT INTO T1 VALUES (b,a);
```

```
END;
```

```
.
```

```
run;
```

Fortuitously, there is only one tuple of T1 that has first component greater than 1, namely (2,4). The

INSERT statement thus inserts (4,2) into T1.

OUTPUT:

```
SQL> CREATE TABLE T1(
  2     e NUMBER(3),
  3     f NUMBER(3));

Table created.

SQL> DELETE FROM T1;

0 rows deleted.

SQL> INSERT INTO T1 VALUES(1, 3);

1 row created.

SQL> INSERT INTO T1 VALUES(2, 4);

1 row created.

SQL> ED
Wrote file afiedt.buf

  1* INSERT INTO T1 VALUES(2, 4)
SQL> ED;
Wrote file afiedt.buf

  1 DECLARE
  2     a NUMBER;
  3     b NUMBER;
  4 BEGIN
  5     SELECT e,f INTO a,b FROM T1 WHERE e>1;
  6     INSERT INTO T1 VALUES(b,a);
  7* END;
SQL> /

PL/SQL procedure successfully completed.

SQL> |
```

```
SQL> ED;  
Wrote file afiedt.buf
```

```
 1 DECLARE  
 2     a NUMBER;  
 3     b NUMBER;  
 4 BEGIN  
 5     SELECT e,f INTO a,b FROM T1 WHERE e>1;  
 6     INSERT INTO T1 VALUES(b,a);  
 7* END;  
SQL> SELECT * FROM T1;
```

E	F
1	3
2	4
4	2

```
SQL> |
```

EXECUTING COMMANDS STORED IN FILE

```
SQL> INSERT INTO T1 VALUES(1,3);
```

```
1 row created.
```

```
SQL> INSERT INTO T1 VALUES(2,4);
```

```
1 row created.
```

```
SQL> @TEST.SQL
```

```
 1 DECLARE  
 2     a NUMBER;  
 3     b NUMBER;  
 4 BEGIN  
 5     SELECT e,f INTO a,b FROM T1 WHERE e>1;  
 6     INSERT INTO T1 VALUES(b,a);  
 7* END;
```

```
PL/SQL procedure successfully completed.
```

```
SQL> SELECT * FROM T1;
```

E	F
1	3
2	4
4	2

```
SQL>
```



RESULT:

Thus the above experiment was successfully completed.

Ex No. 6 Write a PL/SQL block to satisfy some conditions by accepting input from the user.

Date:

Aim: To Write a PL/SQL block to satisfy some conditions by accepting input from the user

Syntax of taking input from the user:

<variablename>:=<variablename>;

Just by writing only this statement we will able to take input from user.

Example:

First write the given code in your SQL command prompt

```
declare
i integer;j
integer; s
integer;
begin
i:=i;----- observe this statement. This statement will tell the machine to take input of i
through user.
j:=j;----- observe this statement. This statement will tell the machine to take input of j
through user.
s:=i+j;
dbms_output.put_line('sum of '||i||' and '||j||' is '||s);
end;
```

Ex No. 7 Write a PL/SQL block that handles all types of exceptions.

Date:

Aim: To a PL/SQL block that handles all types of exceptions

Syntax for Exception Handling

The general syntax for exception handling is as follows. Here you can list down as many exceptions as you can handle. The default exception will be handled using *WHEN others THEN* –

```
DECLARE
    <declarations section>
BEGIN
    <executable command(s)>
EXCEPTION
    <exception handling goes here >
    WHEN exception1 THEN
        exception1-handling-statements
    WHEN exception2 THEN
        exception2-handling-statements
    WHEN exception3 THEN
        exception3-handling-statements
    .....
    WHEN others THEN
        exception3-handling-statements
END;
```

Example

Let us write a code to illustrate the concept. We will be using the CUSTOMERS table we had created and used in the previous chapters –

PL SQL CODE CODE:

```
DECLARE
    c_id customers.id%type := 8;
    c_name customerS.Name%type;
    c_addr customers.address%type;
BEGIN
    SELECT name, address INTO c_name, c_addr
    FROM customers
    WHERE id = c_id;
    DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
    DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);

EXCEPTION
    WHEN no_data_found THEN
        dbms_output.put_line('No such customer!');
    WHEN others THEN
        dbms_output.put_line('Error!');
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result –

No such customer!

PL/SQL procedure successfully completed.

The above program displays the name and address of a customer whose ID is given. Since there is no customer with ID value 8 in our database, the program raises the run-time exception **NO_DATA_FOUND**, which is captured in the **EXCEPTION block**.

Raising Exceptions

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command **RAISE**.

Following is the simple syntax for raising an exception –

```
DECLARE
    exception_name EXCEPTION;
BEGIN
    IF condition THEN
        RAISE exception_name;
    END IF;
EXCEPTION
    WHEN exception_name THEN
        statement;
END;
```

You can use the above syntax in raising the Oracle standard exception or any user-defined exception. In the next section, we will give you an example on raising a user-defined exception. You can raise the Oracle standard exceptions in a similar way.

User-defined Exceptions

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a **RAISE** statement or the procedure **DBMS_STANDARD.RAISE_APPLICATION_ERROR**.

The syntax for declaring an exception is –

```
DECLARE

    my-exception EXCEPTION;
```

Example

The following example illustrates the concept. This program asks for a customer ID, when the user enters an invalid ID, the exception **invalid_id** is raised.

```
DECLARE
    c_id customers.id%type := &cc_id;
    c_name customers.Name%type;
    c_addr customers.address%type;
    -- user defined exception
    ex_invalid_id EXCEPTION;
BEGIN
```

```

IF c_id <= 0 THEN
    RAISE ex_invalid_id;
ELSE
    SELECT name, address INTO c_name, c_addr
    FROM customers
    WHERE id = c_id;
    DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
    DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
END IF;

EXCEPTION
    WHEN ex_invalid_id THEN
        dbms_output.put_line('ID must be greater than zero!');
    WHEN no_data_found THEN
        dbms_output.put_line('No such customer!');
    WHEN others THEN
        dbms_output.put_line('Error!');
END;
/

```

When the above code is executed at the SQL prompt, it produces the following result –

Enter value for cc_id: -6 (let's enter a value -6)

old 2: c_id customers.id%type := &cc_id;

new 2: c_id customers.id%type := -6;

ID must be greater than zero!

PL/SQL procedure successfully completed.

Pre-defined Exceptions

PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO_DATA_FOUND is raised when a SELECT INTO statement returns no rows. The following table lists few of the important pre-defined exceptions –

Exception	Oracle Error	SQLCODE	Description
ACCESS_INTO_NULL	06530	-6530	It is raised when a null object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the choices in the WHEN clause of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to

			assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.
INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when a program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a SELECT INTO statement returns no rows.
NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or sizeconstraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.

Ex. No. 8**Use of Cursors, Procedures and Functions****Date:****Aim:** To demonstrate the use of Cursors, Procedures and Functions**Procedure:**Cursor

1. Declare temporary variables to store the fields of the records.
2. Declare the cursor
3. Open the cursor
4. Start a Loop
5. Fetch the field values of record in the cursor to variables
6. Do the required processing.
7. Update the processed record.
8. Repeat the loop until end of the file is reached
9. Stop

ProceduresProcedure to find smallest of two numbers

1. Declare the required number of variables
2. Create a procedure for finding minimum of two numbers
3. From the main program call the procedure with required parameters.
4. Display the output.

Program Using Cursor

```
DECLARE
  c_regno test.regno%type;
  c_name test.name%type;
  c_mark1 test.mark1%type;
  c_mark2 test.mark2%type;
  i number(2);
  /*type avg IS VARRAY(10) OF number(6,2); */
  c_avg number(6,2);
cursor c_stud is select regno,name,mark1,mark2,avg from test;
BEGIN
  OPEN c_stud;
  -- i:=1;
  LOOP
    FETCH c_stud into c_regno,c_name,c_mark1, c_mark2,c_avg;
    c_avg := (c_mark1 + c_mark2)/2;
    UPDATE test SET avg=c_avg WHERE regno=c_regno;
    EXIT WHEN c_stud%notfound;
  END LOOP;
  CLOSE c_stud;
END;
```

Program Using Procedure

```
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:

```

SQL> ed;
Wrote file afiedt.buf

 1 DECLARE
 2   c_regno test.regno%type;
 3   c_name  test.name%type;
 4   c_mark1 test.mark1%type;
 5   c_mark2 test.mark2%type;
 6   i number(2);
 7   /*type avg IS VARRAY(10) OF number(6,2); */
 8   c_avg number(6,2);
 9   cursor c_stud is select regno,name,mark1,mark2,avg from test;
10 BEGIN
11   OPEN c_stud;
12   -- i:=1;
13   LOOP
14     FETCH c_stud into c_regno,c_name,c_mark1, c_mark2,c_avg;
15     c_avg := (c_mark1 + c_mark2)/2;
16     UPDATE test SET avg=c_avg WHERE regno=c_regno;
17     EXIT WHEN c_stud%notfound;
18   END LOOP;
19   CLOSE c_stud;
20* END;
SQL> /

```

PL/SQL procedure successfully completed.

```
SQL> select * from test;
```

REGNO	NAME	MARK1	MARK2	AVG
5001	ARUN	60	40	50
5040	SANKAR	65	40	52.5

```
SQL>
```

PROCEDURE

```

SQL>
 1 BEGIN
 2   dbms_output.enable;
 3   dbms_output.put_line('This is to test');
 4* END;

```

PL/SQL procedure successfully completed.

```
SQL> set serveroutput on;
```

```
SQL> @ testing.sql;
```

```
This is to test
```

```
This is to test
```

```
This is to test
```

PL/SQL procedure successfully completed.

```
SQL>
```

PROCEDURE

```
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;
/
```

```
SQL> set serveroutput on
SQL> @ proc.sql;
Minimum of (23, 45) : 23
```

```
PL/SQL procedure successfully completed.
```

FUNCTIONS

```
SQL> ed func.sql;

SQL> CREATE OR REPLACE FUNCTION totalstudents
  2 RETURN number IS
  3   total number(2) := 0;
  4 BEGIN
  5   SELECT count(*) into total FROM test;
  6   RETURN total;
  7 END;
  8 /

Function created.

SQL> @ func.sql;

PL/SQL procedure successfully completed.

SQL> set serveroutput on
SQL> @ func.sql;
Total no. of students: 2

PL/SQL procedure successfully completed.

SQL>
```



Ex. No. 9 . Oracle or SQL Server Triggers – Block Level – Form Level Triggers

Date:

Aim: To demonstrate the use of Triggers

Procedure:

1. Create a table named emp with fields for empno, name, department, designation and salary.
2. Create a trigger using CREATE OR REPLACE TRIGGER command.
3. In the trigger write code in such a way that when a new record is inserted or updated or deleted the trigger shoots up and do the following
4. Find difference between existing salary and new salary
5. Display the Old Salary, New Salary and the Difference between old and new Salaries.
6. Insert a record into the table emp and test whether trigger is executed.

PROGRAM

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
sal_diff number;
BEGIN
sal_diff := :NEW.salary - :OLD.salary;
dbms_output.put_line('Old salary: ' || :OLD.salary);
dbms_output.put_line('New salary: ' || :NEW.salary);
dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

OUTPUT:

```
SQL> ALTER TABLE EMP ADD(SALARY NUMBER(10,2));
```

```
Table altered.
```

```
SQL> ED;
```

```
Wrote file afiedt.buf
```

```
 1 CREATE OR REPLACE TRIGGER display_salary_changes BEFORE DELETE OR INSERT OR UPDATE
 2 ON emp
 3 FOR EACH ROW WHEN (NEW.EMPNO > 0)
 4 DECLARE sal_diff number;
 5 BEGIN
 6     sal_diff := :NEW.salary - :OLD.salary;  dbms_output.put_line('Old salary: ' || :OLD.salary);
 7     dbms_output.put_line('New salary: ' || :NEW.salary);
 8     dbms_output.put_line('Salary difference: ' || sal_diff);
 9* END;
```

```
SQL> /
```

```
Trigger created.
```

```
SQL> INSERT INTO EMP VALUES(5055,'KUMAR','MAINTENANCE
 2 ','SUPERVISOR',8389233344,6000);
```

```
Old salary:
```

```
New salary: 6000
```

```
Salary difference:
```

```
1 row created.
```

```
SQL> |
```



Ex. No. 10**Embedded SQL or Database Connectivity****Date:****Aim:** To demonstrate embedded SQL or Database connectivity**Procedure:**

1. Develop database tables in oracle
2. Design the required screen in Visual Basic with all the required tools and objects(text boxes, labels, combo box, option box)
3. Write the coding for connecting the oracle database table with the visual basic application.
4. Run the application.
5. Verify the database connectivity by adding, deleting and viewing records through Visual Basic application.

Program:

```
Dim cnn1 As ADODB.Connection  
Dim rs As ADODB.Recordset  
Dim strcnn As String
```

```
Private Sub ADD_Click()
```

```
With rs
```

```
    .Fields("empname") = nametxt.Text  
    .Fields("dob") = DTPicker1.Value  
    .Fields("gender") = maleopt.Value  
    .Fields("designation") = desgtxt.Value  
    .Fields("dept") = deptcbo.Value  
    .Fields("addr") = addrtxt.Text  
    .Fields("basic") = basictxt.Text  
    .Update
```

```
End With
```

```
    rs.AddNew
```

```
End Sub
```

```
Private Sub cancelcmd_Click()
```

```
    rs.CancelBatch  
    cnn1.CommitTrans
```

```
End Sub
```

```
Private Sub clrcmd_Click()
```

```
    nametxt.Text = ""  
    DTPicker1.Value = ""  
    maleopt.Value = False  
    desgtxt.Value = ""  
    deptcbo.Value = ""  
    addrtxt.Text = ""  
    basictxt.Text = ""
```


End Sub

```
Private Sub delcmd_Click()  
    cnn1.BeginTrans  
    rs.Delete  
    rs.UpdateBatch  
    cnn1.CommitTrans  
    MsgBox ("Record Deleted")  
End Sub
```

```
Private Sub endcmd_Click()  
    End  
End Sub
```

```
Private Sub firstcmd_Click()  
    On Error GoTo l1:  
        rs.Open "Select * from emp1", cnn1, adOpenKeyset, adLockBatchOptimistic  
l1:    rs.MoveFirst  
        transfer
```

End Sub

```
Private Sub Form_Load()  
    Form2.WindowState = 2  
  
    Set cnn1 = New ADODB.Connection  
    Set rs = New ADODB.Recordset  
    rs.CursorLocation = adUseClient  
    'strcnn = "User ID =leo; Password=leo; Data Source = dbserver; Persist Security Info =False"  
    strcnn = "Provider=MSDAORA.1;User ID=leo;Password=leo;Data Source=dbserver;Persist Security  
Info=False"  
    'strcnn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Z:\emp.mdb;Persist Security Info=False"  
    cnn1.Open strcnn
```

End Sub

```
Private Sub lastcmd_Click()  
    On Error GoTo l5:  
        rs.Open "Select * from emp1", cnn1, adOpenKeyset, adLockBatchOptimistic  
l5:    rs.MoveLast  
        transfer
```

End Sub

```
Private Sub modcmd_Click()  
    On Error GoTo l3:  
        rs.Open "Select * from emp1", cnn1, adOpenKeyset, adLockBatchOptimistic  
l3:    cnn1.BeginTrans
```

End Sub

```
Private Sub newcmd_Click()
```

```

On Error GoTo I6:
    rs.Open "Select * from emp1", cnn1, adOpenKeyset, adLockBatchOptimistic

I6: cnn1.BeginTrans
    rs.AddNew

End Sub

Private Sub nextcmd_Click()
On Error GoTo I2:
    rs.Open "Select * from emp1", cnn1, adOpenKeyset, adLockBatchOptimistic
I2:  rs.MoveNext
    transfer
End Sub

Private Sub prevcmd_Click()
On Error GoTo I4:
    rs.Open "Select * from emp1", cnn1, adOpenKeyset, adLockBatchOptimistic
I4:  rs.MovePrevious
    transfer

End Sub

Private Sub savecmd_Click()
With rs
    .Fields("empname") = nametxt.Text
    .Fields("dob") = DTPicker1.Value
    If maleopt.Value = True Then
        .Fields("gender") = "Male"
    Else
        .Fields("gender") = "Female"
    End If
    .Fields("designation") = desgtxt.Text
    .Fields("dept") = deptcbo.Text
    .Fields("addr") = addrtxt.Text
    .Fields("basic") = basictxt.Text
    .UpdateBatch
End With
cnn1.CommitTrans
MsgBox ("Record is saved successfully")

End Sub

Public Sub transfer()
With rs
    If .EOF = False Then
        nametxt.Text = .Fields("empname")
        DTPicker1.Value = .Fields("dob")
    End If
End With

```

```

If .Fields("gender") <> 0 Then
    maleopt.Value = True
Else
    femaleopt.Value = True
End If
desgtxt.Text = .Fields("designation")
deptcbo.Text = .Fields("dept")
addrtxt.Text = .Fields("addr")
basictxt.Text .Fields("basic")
End If
End With

```

End Sub

Output

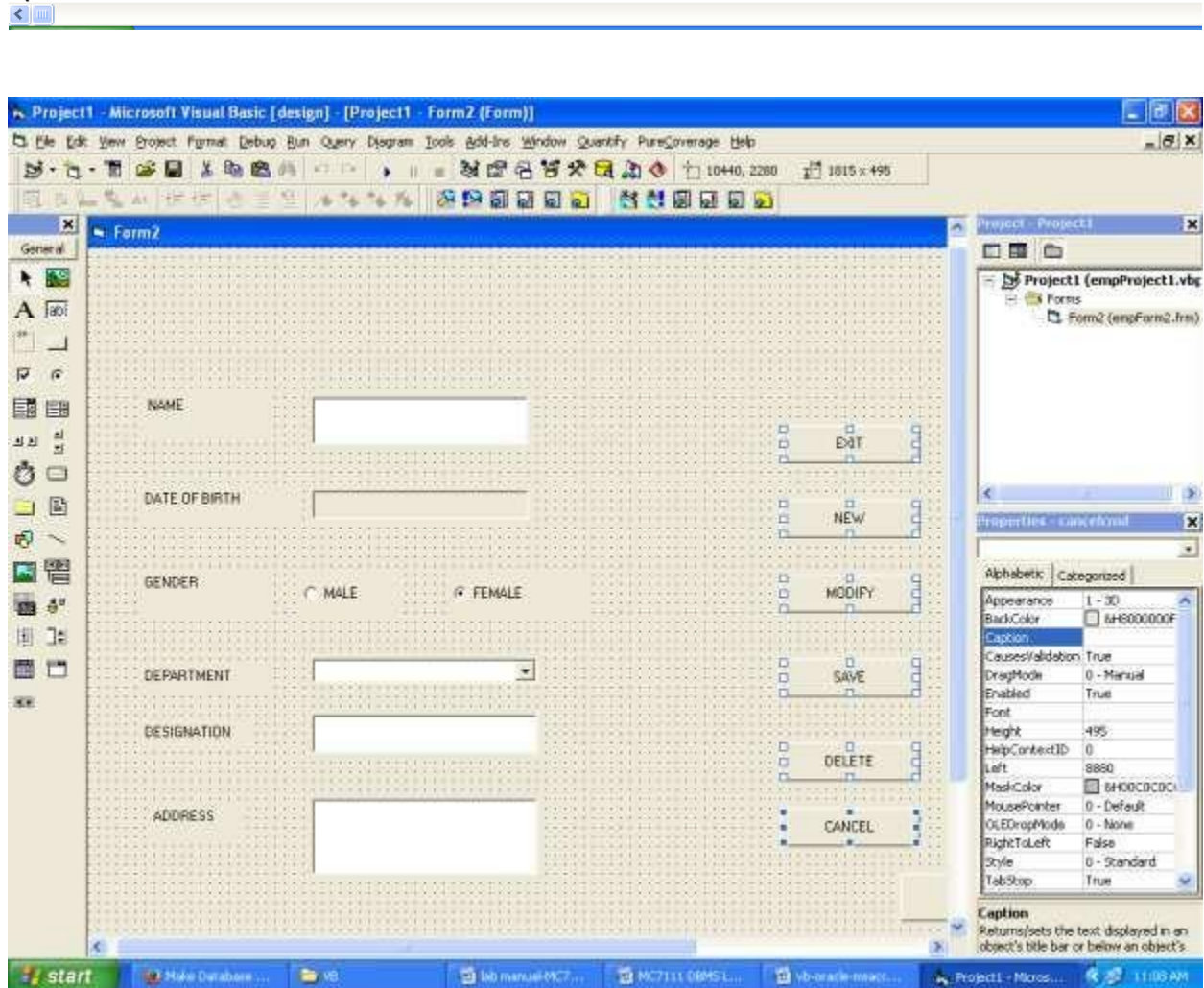
```

1* create table emp1(empname varchar2(25),dob varchar2(10),gender varchar2(6),designation varchar2
SQL> /

```

Table created.

SQL>



```
SQL> select * from emp1;
```

EMPNAME	DOB	GENDER	DESIGNATION
DEPT	ADDR		BASIC
Leo	2/3/2014	True	Technician
B.C.A	Kattankulathur		15000

```
SQL> SELECT * FROM EMP1;
```

EMPNAME	DOB	GENDER	DESIGNATION
DEPT	ADDR		BASIC
Leo	2/3/2014	True	Technician
B.C.A	Kattankulathur		15000
Rani	4/6/2014	Female	TEACHER
MCA	CHENNAI		16000

```
SQL>
```

Ex. No. 11. Front-end tools – Visual Basic/Developer 2000

Date:

Database connectivity using Front End Tools (Application Development using Oracle/ Mysql)

Mini Project

- a) Inventory Control System.
- b) Material Requirement Processing.
- c) Hospital Management System.
- d) Railway Reservation System.
- e) Personal Information System

Aim: To demonstrate embedded SQL or Database connectivity

Procedure:

1. Develop database tables in oracle
2. Design the required screen in Visual Basic with all the required tools and objects(text boxes, labels, combo box, option box)
3. Write the coding for connecting the oracle database table with the visual basic application.
4. Run the application.
5. Verify the database connectivity by adding, deleting and viewing records through Visual Basic application.

Program:

```
Dim cnn1 As ADODB.Connection
```

```
Dim rs As ADODB.Recordset
```

```
Dim strcnn As String
```

```
Private Sub ADD_Click()
```

```
With rs
```

```
    .Fields("sname") = nametxt.Text
```

```
    .Fields("dob") = DTPicker1.Value
```

```
    .Fields("gender") = maleopt.Value
```

```
    .Fields("UG") = ugchk.Value
```

```
    .Fields("PG") = pgchk.Value
```

```
    .Fields("ugcourse") = ugcourse.Text
```

```
    .Fields("pgcourse") = pgcourse.Text
```

```
    .Update
```

```
End With
```

```
    rs.AddNew
```

```
End Sub
```

```
Private Sub cancelcmd_Click()
```

```
    rs.CancelBatch
```

```
    cnn1.CommitTrans
```

```
End Sub
```

```
Private Sub clrcmd_Click()
```

```

    nametxt.Text = " "
    maleopt.Value = True
    femaleopt.Value = True
    ugchk.Value = 0
    pgchk.Value = 0
    ugcourse.Text = ""
    pgcourse.Text = ""
End Sub

Private Sub delcmd_Click()
    cnn1.BeginTrans
    rs.Delete
    rs.UpdateBatch
    cnn1.CommitTrans
    MsgBox ("Record Deleted")
End Sub

Private Sub endcmd_Click()
    End
End Sub

Private Sub firstcmd_Click()
    On Error GoTo l1:
        rs.Open "Select * from personal", cnn1, adOpenKeyset, adLockBatchOptimistic
l1:  rs.MoveFirst
        transfer

End Sub

Private Sub Form_Load()
    Form2.WindowState = 2

    Set cnn1 = New ADODB.Connection
    Set rs = New ADODB.Recordset
    rs.CursorLocation = adUseClient
    strcnn = "User ID =scott; Password=tiger; Data Source = leo; Persist Security Info =False"
' strcnn = "Provider=MSDAORA.1;User ID=scott;Password=tiger;Data
Source=dbserver;Persist Security Info=False"
    strcnn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=E:\student.mdb;Persist Security
Info=False"
    cnn1.Open strcnn

End Sub

Private Sub lastcmd_Click()
    On Error GoTo l5:

```

```

    rs.Open "Select * from personal", cnn1, adOpenKeyset, adLockBatchOptimistic
l5:  rs.MoveLast
    transfer

End Sub

Private Sub modcmd_Click()
On Error GoTo l3:
    rs.Open "Select * from personal", cnn1, adOpenKeyset, adLockBatchOptimistic
l3:  cnn1.BeginTrans

End Sub

Private Sub newcmd_Click()
On Error GoTo l6:
    rs.Open "Select * from personal", cnn1, adOpenKeyset, adLockBatchOptimistic

l6:  cnn1.BeginTrans
    rs.AddNew

End Sub

Private Sub nextcmd_Click()
On Error GoTo l2:
    rs.Open "Select * from personal", cnn1, adOpenKeyset, adLockBatchOptimistic
l2:  rs.MoveNext
    transfer
End Sub

Private Sub pgchk_Click()
If pgchk.Value = 1 Then
pgcourse.Enabled = True
Else
    pgcourse.Enabled = False
End If
End Sub

Private Sub prevcmd_Click()
On Error GoTo l4:
    rs.Open "Select * from personal", cnn1, adOpenKeyset, adLockBatchOptimistic
l4:  rs.MovePrevious
    transfer

End Sub

Private Sub savecmd_Click()
With rs

```

```

.Fields("sname") = nametxt.Text
.Fields("dob") = DTPicker1.Value
.Fields("gender") = maleopt.Value
.Fields("UG") = ugchk.Value
.Fields("PG") = pgchk.Value
.Fields("ugcourse") = ugcourse.Text
.Fields("pgcourse") = pgcourse.Text
.UpdateBatch
End With
cnn1.CommitTrans
MsgBox ("Record is saved successfully")

End Sub

Private Sub ugchk_Click()
If ugchk.Value = 1 Then
ugcourse.Enabled = True
Else
ugcourse.Enabled = False
End If

End Sub

Public Sub transfer()
With rs
If .EOF = False Then
nametxt.Text = .Fields("sname")
DTPicker1.Value = .Fields("dob")
If .Fields("gender") <> 0 Then
maleopt.Value = True
Else
femaleopt.Value = True
End If

ugchk.Value = .Fields("UG")
pgchk.Value = .Fields("PG")
ugcourse.Text = .Fields("ugcourse")
If .Fields("pgcourse") <> "" Then
pgcourse.Text = .Fields("pgcourse")
End If
End If
End With

End Sub

```