KGiSL Institute of Technology - KiTE

Coimbatore – 641 035.

(Approved by AICTE & Affiliated to Anna University, Chennai)

NAME :JOTHI PRASANNA B

REG. No. : 711719104038

SUBJECT : DATABASE MANAGEMENT SYSTEM-CS8481

COURSE : BE COMPUTER SCIENCE AND ENGINNERING

KGiSL Institute of Technology – KiTE

(Approved by AICTE & Affiliated to Anna University, Chennai)
Coimbatore – 641 035.

CS8481 – DataBase Management Systems Laboratory

NAME **:JOTHI PRASANNA B** **CLASS** **:** II CSE-A

UNIVERSITY REG NO : 711719104038

Certified that, this is a bonafide record of work done by **JOTHI PRASANNA B** of **Computer Science and Engineering** branch in **DataBase Management Systems Laboratory**, during **Fourth** semester of academic year 2020- 2021.

Faculty In-charge **Head of the Department**

Submitted during Anna University Practical Examination held on at KGiSL Institute of Technology, Coimbatore – 641 035.

Internal Examiner **External Examiner**

INDEX

S. No	DATE	LIST OF THE EXPERIMENTS	PAGE NO	MARKS	SIGNATURE
1.		CREATION OF A DATABASE AND WRITING SQL QUERIES TO RETRIEVE INFORMATION FROM THE DATABASE			
2.		DATABASE QUERYING - SIMPLE QUERIES , NESTED , SUB QUERIES AND JOINS			
3.		CREATION OF VIEWS, SYNONYMS, SEQUENCE, INDEXES, SAVE POINT			
4.		DATABASE PROGRAMMING:IMPLICIT AND EXPLICIT CURSORS			
5.		CREATION OF PROCEDURE			
6.		CREATION OF DATABASE TRIGGERS			
7.		WRITE A PL/SQL BLOCK THAT HANDLES ALL TYPES OF EXCEPTIONS			
8.		DATABASE DESIGN USING E-R MODEL AND NORMALIZATION			
9.		DESIGN AND IMPLEMENTATION OF BANKING SYSTEM			
10.		AUTOMATIC BACKUP OF FILES AND RECOVERY			

Vision and Mission of KGiSL Institute of Technology

En Vision ed Future - ***"More genius per genius"***

- To be recognized as the **#1** engineering institution regionally and nationally by all stakeholders, including employers, faculty and society.

Core Mission Question - how can we maximize learner transformation in 10,440 hours?

- We are co-responsible for producing remarkable behavioral traits such as deep inquiry (self generated questions, curiosity, research),
- an intrinsic desire for uncomfortable struggle (for employable skills, specific interests, big ideas) and
- an inclusive mindset (real-world projects, collaboration, compassion)

Vision and Mission of Department of Computer Science and Engineering

Vision

To promote industry embedded education there by creating Computer Science Professionals with exceptional intellectual skills that has a transformative impact on society.

Mission

- To inculcate a remarkable behavioral traits and industry embedded research, leading to face uncomfortable struggle
- To foster the spirit of deep enquiry and imagination among students by bringing the curiosity to come up with innovative ideas for well-being of the society
- To fasten with individuals and organizations for realizing supreme potential for solving real-world problems

Programme: B.E. Computer Science and Engineering

PROGRAM EDUCATIONAL OBJECTIVES (PEOs):

PEO1: To enable graduates to pursue higher education and research, or have a successful career in industries associated with Computer Science and Engineering, or as entrepreneurs.
PEO2: To ensure that graduates will have the ability and attitude to adapt to emerging technological changes.
PEO3: To attain professional skills by ensuring life-long learning with a sense of social values.

PROGRAM OUTCOMES POs:

At the time of graduation, the students of Computer Science and Engineering should have the

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: 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 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.

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 ENVIRONMENT & 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 INDIVIDUAL AND TEAM WORK: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 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.

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.

PROGRAM SPECIFIC OUTCOMES (PSO’s)

PSO1: To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO2: To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO3: To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems.

EX.NO:1 DATE :	CREATION OF A DATABASE AND WRITING SQL QUERIES4TO RETRIEVE INFORMATION FROM THE DATABASE

AIM

To create table and write SQL queries to retrieve information from the database using MySQL

CREATE TABLE

This command is used to create the structure of a new table

Syntax:

CREATE TABLE TABLE_NAME(FIELD1 DATATYPE(SIZE),…….,FIELDn DATATYPE(SIZE));

Query and output:

```
SQL> CREATE TABLE EMPLOYEE(EMP_ID INT,EMP_NAME VARCHAR(10),EMP_ADDRESS VARCHAR(10));

Table created.
```

TO SHOW THE TABLE DEFINITION STRUCTURE

This command is used to provide information about the columns in a table

Syntax:

DESC TABLE_NAME;

Query and output:

```
SQL> DESC EMPLOYEE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMP_ID                                             NUMBER(38)
 EMP_NAME                                           VARCHAR2(10)
 EMP_ADDRESS                                        VARCHAR2(10)
```

TO ADD COLUMN TO THE TABLE

This command is used to add some extra xolumns into an existing table

Syntax:

ALTER TABLE TABLE_NAME ADD COLUMN_NAME DATATYPE(SIZE);

Query and output:

```
SQL> ALTER TABLE EMPLOYEE ADD EMP_SALARY INT;

Table altered.
```

RENAMING A TABLE

This command is used to rename a table provided you are the owner of the table

Syntax:

ALTER TABLE OLD_NAME RENAME TO NEW_NAME;

Query and output:

```
SQL> ALTER TABLE EMPLOYEE RENAME TO EMPLOY;

Table altered.
```

TRUNCATE

Truncating a table is removing all records from the table. The structure of the table stays intact

Syntax:

TRUNCATE TABLE TABLE_NAME;

Query and output:

```
SQL> TRUNCATE TABLE EMPLOY;

Table truncated.
```

DROP TABLE

This command is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table. Once a table is deleted then all the information available in the table would also be lost forever.

Syntax:

DROP TABLE TABLE_NAME;

Query and output:

```
SQL> DROP TABLE EMPLOY;

Table dropped.
```

CREATE TABLE

This command is used to create the structure of a new table

Syntax:

CREATE TABLE TABLE_NAME(FIELD1 DATATYPE(SIZE),…….,FIELDn DATATYPE(SIZE));

Query and output:

```
SQL> CREATE TABLE EMPLOYEE(EMP_ID INT,EMP_NAME VARCHAR(10),EMP_ADDRESS VARCHAR(10));

Table created.
```

INSERT ROWS INTO THE TABLE

This command is used to insert rows into a table

Syntax:

INSERT INTO TABLE_NAME VALUES (VALUE1, VALUE2,…., VALUEN);

Query and output:

```
SQL> INSERT INTO EMPLOYEE VALUES(1,'SAM','COIMBATORE');

1 row created.



SQL> INSERT INTO EMPLOYEE VALUES(2,'ASHMITHA','CHENNAI');
SQL> INSERT INTO EMPLOYEE VALUES(3,'RAM','OOTY');

1 row created.
```

TO DISPLAY THE TABLE WITH VALUES

This command is used to display table with all its values

Syntax:

SELECT * FROM TABLE_NAME;

Query and output:

```
SQL> SELECT * FROM EMPLOYEE;

    EMP_ID EMP_NAME   EMP_ADDRES
---------- ---------- ----------
         1 SAM        COIMBATORE
         2 ASHMITHA   CHENNAI
         3 RAM        OOTY
```

COMMIT

This command is used to permanently save any transaction into database

Syntax:

COMMIT;

Query and output:

```
SQL> COMMIT;

Commit complete.
```

TO DISPLAY A PARTICULAR ROW

This command is used display a particular row from a table

Syntax:

SELECT * FROM TABLE_NAME WHERE CONDITION;

Query and output:

```
SQL> SELECT * FROM EMPLOYEE WHERE EMP_ID=1;

    EMP_ID EMP_NAME   EMP_ADDRES
---------- ---------- ----------
         1 SAM        COIMBATORE
```

TO DELETE A PARTICULAR ROW

This command is used to delete a particular row

Syntax:

DELETE FROM TABLE_NAME WHERE CONDITION;

Query and output:

```
SQL> DELETE FROM EMPLOYEE WHERE EMP_ID=3;

1 row deleted.
```

ROLLBACK

This command restores database to last command state

Syntax:

ROLLBACK;

Query and output:

```
SQL> ROLLBACK;

Rollback complete.
```

CREATE TABLE

This command is used to create the structure of a new table

Syntax:

CREATE TABLE TABLE_NAME(FIELD1 DATATYPE(SIZE),…….,FIELDn DATATYPE(SIZE));

Query and output:

```
SQL> CREATE TABLE STUDENT (ROLL_NO INT,NAME VARCHAR(10),DEPT VARCHAR(10));

Table created.
```

INSERT ROWS INTO THE TABLE

This command is used to insert rows into a table

Syntax:

INSERT INTO TABLE_NAME VALUES (VALUE1, VALUE2,…., VALUEN);

Query and output:

```
SQL> INSERT INTO STUDENT VALUES(1,'ACHU','CSE');

1 row created.
SQL> INSERT INTO STUDENT VALUES(2,'RAJ','MECH');

1 row created.




SQL> INSERT INTO STUDENT VALUES(3,'SUVI','CSE');

1 row created.
```

TO DISPLAY THE TABLE WITH VALUES

This command is used to display table with all its values

Syntax:

SELECT * FROM TABLE_NAME;

Query and output:

```
SQL> SELECT * FROM STUDENT;

   ROLL_NO NAME       DEPT
---------- ---------- ----------
         1 ACHU       CSE
         2 RAJ        MECH
         3 SUVI       CSE
```

TO UPDATE A VALUE

This command is used to update a particular value in a table

Syntax:

UPDATE TABLE_NAME NEW_VALUE WHERE CONDITION;

Query and output:

```
SQL> UPDATE STUDENT SET DEPT='ECE' WHERE ROLL_NO=1;

1 row updated.
```

TO DISPLAY THE TABLE WITH VALUES

This command is used to display table with all its values

Syntax:

SELECT * FROM TABLE_NAME;

Query and output:

```
SQL> SELECT * FROM STUDENT;

   ROLL_NO NAME       DEPT
---------- ---------- ----------
         1 ACHU       ECE
         2 RAJ        MECH
         3 SUVI       CSE
```

SAVEPOINT

This command is used to temporarily save a transaction so that you can rollback to the point whenever necessary

Syntax:

SAVEPOINT SAVEPOINT_NAME;

Query and output:

```
SQL> SAVEPOINT S1;

Savepoint created.
```

DELETE ALL ROWS FROM THE TABLE

This command is used to delete all values from the table

Syntax:

DELETE FROM TABLE_NAME;

Query and output:

```
SQL> DELETE FROM STUDENT;

3 rows deleted.
```

ROLLBACK

This command restores database to last command state

Syntax:

ROLLBACK;

Query and output:

```
SQL> ROLLBACK TO S1;

Rollback complete.
```

TO DISPLAY THE TABLE WITH VALUES

This command is used to display table with all its value

Syntax:

SELECT * FROM TABLE_NAME;

Query and output:

```
SQL> SELECT * FROM STUDENT;

   ROLL_NO NAME       DEPT
---------- ---------- ----------
         1 ACHU       ECE
         2 RAJ        MECH
         3 SUVI       CSE
```

GRANT

Grant is a command used to access or privileges on the database objects to the users

Syntax:

GRANT PRIVILEGE_NAME ON OBJECT_NAME TO {USER_NAME|PUBLIC|ROLENAME};

Query and output:

```
SQL> GRANT CREATE TABLE TO STU;
```

REVOKE

The revoke command removes user access rights or privileges to the database objects

Syntax:

REVOKE PRIVILEGE_NAME ON OBJECT_NAME FROM {USER_NAME |PUBLIC | ROLE_NAME};

Query and output:

```
SQL> REVOKE CREATE TABLE TO STU;
```

RESULT

The SQL queries to retrieve information from the database using MySQL is executed successfully.

EX.NO : 2 DATE :	DATABASE QUERYING - SIMPLE QUERIES , NESTED , SUB QUERIES AND JOINS

AIM:

To create table and write SQL queries to retrieve information from the database using MySQL

CREATE TABLE

This command is used to create the structure of a new table

Syntax:

CREATE TABLE TABLE_NAME(FIELD1 DATATYPE(SIZE),…….,FIELDn DATATYPE(SIZE));

Query and output:

```
SQL> CREATE TABLE STU (PER_ROLL INT,STU_NAME VARCHAR(10),PHONE INT,ADDRESS VARCHAR(20),EMAIL VARCHAR
(20));

Table created.
```

INSERT ROWS INTO THE TABLE

This command is used to insert rows into a table

Syntax:

INSERT INTO TABLE_NAME VALUES (VALUE1, VALUE2,…., VALUEN);

Query and output:

```
SQL> INSERT INTO STU VALUES(1,'ACHU',12345,'COIMBATORE','A@GMAIL.COM');

1 row created.

SQL> INSERT INTO STU VALUES(2,'BAVI',23456,'CHENNAI','B@GMAIL.COM');

1 row created.
SQL> INSERT INTO STU VALUES(4,'DHANU',45678,'OOTY','D@GMAIL.COM');

1 row created.

SQL> INSERT INTO STU VALUES(3,'CHINU',34567,'BANGALORE','C@GMAIL.COM');

1 row created.
```

TO DISPLAY THE TABLE WITH VALUES

This command is used to display table with all its values

Syntax:

SELECT * FROM TABLE_NAME;

Query and output:

Query and output:

```
SQL> SELECT * FROM STU;

  PER_ROLL STU_NAME        PHONE ADDRESS              EMAIL
---------- ---------- ---------- -------------------- --------------------
         1 ACHU            12345 COIMBATORE           A@GMAIL.COM
         2 BAVI            23456 CHENNAI              B@GMAIL.COM
         3 CHINU           34567 BANGALORE            C@GMAIL.COM
         4 DHANU           45678 OOTY                 D@GMAIL.COM
```

CREATE TABLE

This command is used to create the structure of a new table

Syntax:

CREATE TABLE TABLE_NAME(FIELD1 DATATYPE(SIZE),…….,FIELDn DATATYPE(SIZE));

Query and output:

```
SQL> CREATE TABLE PROJECT(PRO_ROLL INT,NAME VARCHAR(10),INTEREST VARCHAR(10));

Table created.
```

INSERT ROWS INTO THE TABLE

This command is used to insert rows into a table

Syntax:

INSERT INTO TABLE_NAME VALUES (VALUE1, VALUE2,…., VALUEN);

Query and output:

```
SQL> INSERT INTO PROJECT VALUES(1,'ACHU','AI');

1 row created.

SQL> INSERT INTO PROJECT VALUES(3,'CHINU','ML');

1 row created.


SQL> INSERT INTO PROJECT VALUES(4,'DHANU','RPA');

1 row created.
```

TO DISPLAY THE TABLE WITH VALUES

This command is used to display table with all its values

Syntax:

SELECT * FROM TABLE_NAME;

```
SQL> SELECT * FROM PROJECT;

  PRO_ROLL NAME       INTEREST
---------- ---------- ----------
         1 ACHU       AI
         3 CHINU      ML
         4 DHANU      RPA
```

TO DISPLAY A PARTICULAR ROW

This command is used display a particular row from a table

Syntax:

SELECT COLUMN_NAMES FROM TABLE_NAME WHERE CONDITION;

Query and output:

```
SQL> SELECT PER_ROLL,STU_NAME,INTEREST FROM STU,PROJECT WHERE PER_ROLL=PRO_ROLL;

  PER_ROLL STU_NAME   INTEREST
---------- ---------- ----------
         1 ACHU       AI
         3 CHINU      ML
         4 DHANU      RPA
```

TO PERFORM INNER JOIN

This command is used to select records that have matching values in both tables

Syntax:

SELECT COLUMN_NAME(S) FROM TABLE1 INNER JOIN TABLE2 ON TABLE1.COLUMN_NAME=TABLE2.COLUMN_NAME;

Query and output:

```
SQL> SELECT STU.PER_ROLL,STU.STU_NAME,PROJECT.INTEREST FROM STU INNER JOIN PROJECT ON STU.PER_ROLL=P
RO_ROLL;

  PER_ROLL STU_NAME   INTEREST
---------- ---------- ----------
         1 ACHU       AI
         3 CHINU      ML
         4 DHANU      RPA
```

TO PERFORM RIGHT OUTER JOIN

This command is used to select all records from right table and the matched records from the left table

Syntax:

SELECT COLUMN_NAME(S) FROM TABLE1 RIGHT OUTER JOIN TABLE2 ON TABLE1.COLUMN_NAME=TABLE2.COLUMN_NAME;

```
SQL> SELECT STU.PER_ROLL,STU.STU_NAME,PROJECT.INTEREST FROM STU RIGHT OUTER JOIN PROJECT ON STU.PER_
ROLL=PRO_ROLL;

  PER_ROLL STU_NAME   INTEREST
---------- ---------- ----------
         1 ACHU       AI
         3 CHINU      ML
         4 DHANU      RPA
```

TO PERFORM LEFT OUTER JOIN

This command is used to select all records from left table and the matched records from the right table

Syntax:

SELECT COLUMN_NAME(S) FROM TABLE1 LEFT OUTER JOIN TABLE2 ON TABLE1.COLUMN_NAME=TABLE2.COLUMN_NAME;

Query and output:

```
SQL> SELECT STU.PER_ROLL,STU.STU_NAME,PROJECT.INTEREST FROM STU LEFT OUTER JOIN PROJECT ON STU.PER_R
OLL=PRO_ROLL;

  PER_ROLL STU_NAME   INTEREST
---------- ---------- ----------
         1 ACHU       AI
         3 CHINU      ML
         4 DHANU      RPA
         2 BAVI
```

TO PERFORM FULL JOIN

This command is used to select records that have matching values in either left or right table

Syntax:

SELECT COLUMN_NAME(S) FROM TABLE1 FULL JOIN TABLE2 ON TABLE1.COLUMN_NAME=TABLE2.COLUMN_NAME;

Query and output:

```
SQL> SELECT STU.PER_ROLL,STU.STU_NAME,PROJECT.INTEREST FROM STU FULL JOIN PROJECT ON STU.PER_ROLL=PR
O_ROLL;

  PER_ROLL STU_NAME   INTEREST
---------- ---------- ----------
         1 ACHU       AI
         3 CHINU      ML
         4 DHANU      RPA
         2 BAVI
```

TO PERFORM CROSS JOIN

This command is used to generate a paired combination of each row of the first table with each row of the second table

Syntax:

SELECT * FROM TABLE1 CROSS JOIN TABLE2;

```
SQL> SELECT * FROM STU CROSS JOIN PROJECT;

  PER_ROLL STU_NAME        PHONE ADDRESS              EMAIL
---------- ---------- ---------- -------------------- --------------------
  PRO_ROLL NAME       INTEREST
---------- ---------- ----------
         1 ACHU             12345 COIMBATORE           A@GMAIL.COM
         1 ACHU       AI

         2 BAVI             23456 CHENNAI              B@GMAIL.COM
         1 ACHU       AI

         3 CHINU            34567 BANGALORE            C@GMAIL.COM
         1 ACHU       AI


  PER_ROLL STU_NAME        PHONE ADDRESS              EMAIL
---------- ---------- ---------- -------------------- --------------------
  PRO_ROLL NAME       INTEREST
---------- ---------- ----------
         4 DHANU            45678 OOTY                 D@GMAIL.COM
         1 ACHU       AI

         1 ACHU             12345 COIMBATORE           A@GMAIL.COM
         3 CHINU      ML

         2 BAVI             23456 CHENNAI              B@GMAIL.COM
         3 CHINU      ML


  PER_ROLL STU_NAME        PHONE ADDRESS              EMAIL
---------- ---------- ---------- -------------------- --------------------
  PRO_ROLL NAME       INTEREST
---------- ---------- ----------
         3 CHINU            34567 BANGALORE            C@GMAIL.COM
         3 CHINU      ML

         4 DHANU            45678 OOTY                 D@GMAIL.COM
         3 CHINU      ML

         1 ACHU             12345 COIMBATORE           A@GMAIL.COM
         4 DHANU      RPA


  PER_ROLL STU_NAME        PHONE ADDRESS              EMAIL
---------- ---------- ---------- -------------------- --------------------
  PRO_ROLL NAME       INTEREST
---------- ---------- ----------
         2 BAVI             23456 CHENNAI              B@GMAIL.COM
         4 DHANU      RPA

         3 CHINU            34567 BANGALORE            C@GMAIL.COM
         4 DHANU      RPA

         4 DHANU            45678 OOTY                 D@GMAIL.COM
         4 DHANU      RPA


12 rows selected.
```

SIMPLE QUERY CREATE TABLE:

This command is used to create the structure of a new table

Syntax:

CREATE TABLE TABLE_NAME(FIELD1 DATATYPE(SIZE),…….,FIELDn DATATYPE(SIZE));

Query and output:

```
SQL> CREATE TABLE EMPLOYEE(EMP_ID INT,EMP_NAME VARCHAR(10),PHONE_NUM INT,ADDRESS VARCHAR(20));

Table created.
```

INSERT ROWS INTO THE TABLE

This command is used to insert rows into a table

Syntax:

INSERT INTO TABLE_NAME VALUES (VALUE1, VALUE2,…., VALUEN);

Query and output:

```
SQL> INSERT INTO EMPLOYEE VALUES(1,'ACHU',12345,'COIMBATORE');

1 row created.

SQL> INSERT INTO EMPLOYEE VALUES(2,'BAVI',23456,'CHENNAI');

1 row created.

SQL> INSERT INTO EMPLOYEE VALUES(3,'CHINU',34567,'BANGALORE');

1 row created.
```

TO DISPLAY THE TABLE WITH VALUES

This command is used to display table with all its values

Syntax:

SELECT * FROM TABLE_NAME;

Query and output:

```
SQL> SELECT * FROM EMPLOYEE;

    EMP_ID EMP_NAME    PHONE_NUM ADDRESS
---------- ---------- ---------- --------------------
         1 ACHU            12345 COIMBATORE
         2 BAVI            23456 CHENNAI
         3 CHINU           34567 BANGALORE
```

SUBQUERY CREATE TABLE:

This command is used to create the structure of a new table

```
SQL> CREATE TABLE EMPLOYEE_SALARY(ID INT,NAME VARCHAR(10),SALARY INT);

Table created.
```

INSERT ROWS INTO THE TABLE

This command is used to insert rows into a table

Syntax:

INSERT INTO TABLE_NAME VALUES (VALUE1, VALUE2,…., VALUEN);

Query and output:

```
SQL> INSERT INTO EMPLOYEE_SALARY VALUES(1,'ACHU',25000);

1 row created.

SQL> INSERT INTO EMPLOYEE_SALARY VALUES(2,'BAVI',35000);

1 row created.

SQL> INSERT INTO EMPLOYEE_SALARY VALUES(3,'CHINU',45000);

1 row created.
```

TO DISPLAY THE TABLE WITH VALUES

This command is used to display table with all its values

Syntax:

SELECT * FROM TABLE_NAME;

Query and output:

```
SQL> SELECT * FROM EMPLOYEE_SALARY;

        ID NAME            SALARY
---------- ---------- ----------
         1 ACHU            25000
         2 BAVI            35000
         3 CHINU           45000
```

TO DISPLAY A PARTICULAR ROW

This command is used display a particular row from a table

Syntax:

SELECT * FROM TABLE_NAME WHERE CONDITION;

Query and output:

```
SQL> SELECT * FROM EMPLOYEE WHERE EMP_ID IN(SELECT ID FROM EMPLOYEE_SALARY WHERE SALARY>30000);

    EMP_ID EMP_NAME    PHONE_NUM ADDRESS
---------- ---------- ---------- --------------------
         2 BAVI            23456 CHENNAI
         3 CHINU           34567 BANGALORE
```

RESULT

The SQL queries to retrieve information from the database using MySQL is executed successfully

Ex. No: 3 Date:	CREATION OF VIEWS,SYNONYMS,SEQUENCE,INDEXES,SAVE POINT

AIM:

Creation of Views, Synonyms, Sequence, Indexes, save point from the database using MySQL.

1. VIEW

A view is object that gives the user a logical view of data from underlying tables.

Creating a table book to display the book details.

SQL>select*from book;

ISBN	TITLE	AUTHOR	QUANTITY	PRICE	PUB_YEAR
1001	DBMS	SEEMA	321	215	2000
1002	COA	JOHN	350	400	2012

1. Create of views.

Syntax:

Create view view_name as select column name 1,column name 2 from table name where column name=expression list;

Query:

create view v_book as select Title, author from book;

View created.

Output:

TITLE	AUTHOR
DBMS	SEEMA
COA	JOHN

2. Select Data from View

Query:

Select title from v_book where author=’SEEMA’;

Output:

TITLE
DBMS

3. Drop View

Syntax:

Drop view view name;

Query:

Drop view v_book;

Output:

View dropped.

2. SYNONYMS

1. Synonyms

A synonym is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects.

Create Synonym (Or Replace)

Create synonym customers for store.customers;

2. Public Synonyms

create public synonym products for schemaname.tablename;

3. Creating a synonym for a table

create table product (product_name varchar2(25) primary key, product_price number(4,2), quantity _on_hand number(5,0), last_stock_date date);

Output

Table created.

sql> **insert** into product values ('product 1', 99, 1, '15-jan-03');

Output

1 row created.

sql> select * from prod;

Output

select * from prod
ERROR at line 1:
ORA-00942: table or view does not exist

sql> create synonym prod for product;

Output

synonym created.

sql> **select** * **from** prod;

Output

PRODUCT_NAME	PRODUCT_PRICE	QUANTITY_ON_HAND	LAST_STOC
Product 1	99	1	15-JAN-03
Product 2	75	1000	15-JAN-02

6 rows selected.

sql> drop synonym prod;

Output:

Synonym dropped.

sql> drop table product;

Table dropped.

4. Creating a public synonym

create table product (product_name varchar2(25) primary key, product_price number(4,2), quantity_on_hand number(5,0), last_stock_date date);

Output

Table created.
sql> **insert** into product values ('product 1', 99, 1,   '15-jan-03');

Output
1 row created.
sql> **create** public synonym product for product;
Output
synonym created.
sql> drop public synonym product;
Output
Synonym dropped.
sql> drop table product;
Output
Table dropped.

5. Create a synonym for a view

create table emp(emp_id integer primary key, lastname varchar2(20) not null, firstname varchar2(15) not null, midinit varchar2(1), street varchar2(30), city varchar2(20) varchar2(2), zip varchar2(5), shortZipCode varchar2(4), area_code varchar2(3), phone varchar2(8), company_name varchar2(50));
Output
Table created.
sql> insert into emp(emp_id,lastname,firstname,midinit,street,city,state,zip,shortZipCode,area_code, phone,company_name)values(1,'Jones','Joe','J','1 Ave','New York','NY','11202','1111','212', '221-4333', 'Big Company');
Output
1 row created.
sql>create or replace view phone_list as select emp_id, firstname || ' ' || midinit || '. ' || lastname as na me,'(' || area_code || ')' || phone as telephone# from emp;
Output
View created.
sql> desc phone_list
Output

NAME	NULL?	TYPE
EMP_ID	NOT NULL	NUMBER(38)
NAME		VARCHAR2(39)
TELEPHONE		VARCHAR2(13)

sql> select * from phone_list;

Output

EMP_ID	NAME	TELEPHONE#
1	Joe J. Jones	(212)221-4333
2	Sue J. Smith	(212)436-6773

3 rows selected.
sql> create synonym phones for phone_list;

Output

Synonym created.

sql> desc phones;

Output

NAME	NULL?	TYPE
EMP_ID	NOT NULL	NUMBER(38)
NAME		VARCHAR2(39)
TELEPHONE#		VARCHAR2(13)

sql> **select** * **from** phones;

Output

EMP_ID	NAME	TELEPHONE#
1	Joe J. Jones	(212)221-4333
2	Sue J. Smith	(212)436-6773
3	Peggy J. X	(212)234-4444

3 rows selected.

sql> select view_name from user_views;

Output

VIEW_NAME
EMP_HQ
AVG_SAL
EMPDEPT_V
DEPT_SAL
ALL_ORACLE_ERRORS
INVENTORY_VIE
TOP_EMP
EMP_BONUS
SHARED
PHONE_LIST

10 rows selected.

sql> select synonym_name, table_name from user_synonyms;

Output

SYNONYM_NAME	TABLE_NAME
PHONES	PHONE_LIST

1 row selected.

6. .Dropping a synonym

sql> drop synonym phones ;

Output

synonym dropped.

sql> drop table emp;

Output

Table dropped.

7. Dropping a public synonym

create public synonym product for product;

Output

synonym created.
sql> drop public synonym product;

Output
synonym dropped.
SQL> drop table product;
Output
Table dropped.

8. Query a table by selecting its synonym

create table employees(empno number(4), ename varchar2(8), init varchar2(5), job varchar2(8), mgr number(4), bdate date , msal number(6,2), comm number(6,2), deptno number(2)) ;
Output

Table created.
SQL> create synonym e for employees;
Output
Synonym created.
sqL> select * from e;

EMPNO	ENAME	INIT	JOB	MGR	BDATE	MSAL	DEPTNO
1	JASON	N	TRAINER	2	18-DEC	800	10
2	JERRY	J	SALESREP	3	19-NOV	1600	10
3	JORD	T	SALESREP	5	21-OCT	500	10

sql> drop table employees;
Output

Table dropped.
sql> drop synonym e;
Output
Synonym dropped.

9. Seeing SYNONYM in the Oracle data dictionary
create public synonym product for product;

Output
synonym created.
sql> select table_name, substr(comments, 1, 45) from dict where substr(comments, 1, 7) <> 'synonym' and rownum < 50;
sql> drop public synonym product;

Output
synonym dropped.
sql> drop table product;
Output

table dropped.

10. Describe a synonym

create table employees(empno number(4), ename varchar2(8), init varchar2(5), job varchar2(8), mgr number(4), bdate date, msal number(6,2), comm number(6,2), deptno number(2)) ;

Output

table created.

sql> insert into employees values(1,'jason', 'n', 'trainer', 2, date '1965-12-18', 800 , null, 10);

Output

1 row created.

create synonym e for employees;

Output

Synonym created.

SQL> describe e;

Output

NAME	NULL	TYPE
EMPNO		NUMBER(4)
ENAME		VARCHAR2(8)
INIT		VARCHAR2(5)
JOB		VARCHAR2(8)
MGR		NUMBER(4)
BDATE		DATE
MSAL		NUMBER(6,2)
COMM		NUMBER(6,2)
DEPTNO		NUMBER(2)

sql> drop table employees;

Output

table dropped.

sql> drop synonym e;

Output

synonym dropped.

3. SEQUENCE

A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value, and sequences provide an easy way to generate them.

Syntax

Create sequemce sequence-name
Start with initial-value
Increment by increment-value
Maxvalue maximum-value
Cycle|nocycle;

Explanation

Initial-value specifies the starting value of sequence, increment-value is the value by which sequence will be incremented, Maxvalue specifies the maximum value which sequence will increment itself. Cycle specifies that if the maximum value exceeds the set limit, sequence will

restart its cycle from the beginning. No cycle specifies that if sequence exceeds maxvalue an error will be thrown.

i. Create Sequence
Query:
create sequency seq_1
Minvalue 1
Maxvalue 80
Startwith 10
Increament by1
Cache 10;
Sequence created
ii. View the Next Value:
Syntax
select sequence_name.nextual from dual;
Query
select seq_1.nextval from dual;
Output
nextvalue
NEXTVALUE
11

iii. Alter Sequence:
Syntax
alter sequency sequence_name increament by value;
Query
alter sequency seq_1 increament by 5;
Sequence altered

iv. View The Current Value:

Syntax
select sequence_name.current from dual;
Query
select seq_1.currval from dual;
Output
current value
CURRVAL
11

i. Create a table:
Create table test(rec_id number,rec_text varchar(15));
Table created.

ii. Create sequence:
Syntax
create sequence sequence_name;
Query
create a sequence seq_2;
Sequence created.

iii. View The Next Value:

Syntax

Select sequence_name.nextval from dual;

Query

select seq_2.nextval from dual;

Output

nextvalue

NEXTVAL

12

v. Insert Values:

Syntax

insert into table name value(sequence name.nextval,field name);

Query

insert into test values(seq_2.naextval 'Record A');

insert into test values(seq_2.naextval 'Record B');

output:

REC_ID	REC_TEXT
13	Record a
14	Record b

Drop Sequence

Drops a sequence generator previously created with the create sequence statement. A sequence generator may be changed by dropping the sequence and then recreating it.

Syntax

drop sequence name;

Query

Drop sequence seq_2;

Sequence dropped

3. INDEX

Indexes allow the database application to find data fast; without reading the whole table

1. Creating an Index

create table employee(id varchar2(4 byte) not null, first_name varchar2(10), last_name varchar2(10),
start_date date, end_date date, salary number(8,2), city varchar2(10), description varchar2(15));

Output

Table created.

sql> insert into employee(id, first_name, last_name, start_date, end_date, salary, city, description) values ('01','jason', 'martin', to_date('19960725','yyyymmdd'), to_date('20060725','yyyymmdd'), 12 34.56, 'toronto', 'programmer');

Output

1 row created.

select * from employee;

Output

ID	FIRST_NAME	LAST_NAME	START_DAT	END_DATE	SALARY	CITY	DESCRIPTION
1	Jason	Martin	25-jul-96	25-jul-06	1234.56	Toronto	Programmer
2	Alison	Mathew	21-mar-76	21-feb-86	6661.78	Vancouver	Tester
3	Jmaes	Smith	12-dec-78	15-mar-90	6544.78	Vancouver	Tester
4	Celia	Rice	24-oct-82	21-apr-99	2344.78	Vancouver	Manager
5	Robert	Black	15-jan-84	08-aug-98	2334.78	Vancouver	Tester

5 rows selected.
sql> create index employee_last_name_idx on employee(last_name);

Output
index created.
sql> drop index employee_last_name_idx;

Output
index dropped.
sql> drop table employee;

table dropped.

2. Enforce uniqueness of values in a column using a unique index
create table employee…….
sql> create unique index employee_id_idx on employee(id);

Output
index created.
sql> drop index employee_id_idx;

Output
index dropped.

3. Create combined-column index
create table person…….
create index person_name_index on person(last_name, first_name);
Output
index created.
sql> drop index person_name_index;

Output
index dropped.

4. Create a composite index on multiple columns
Create table Employee……
sql >create index employee_first_last_name_idx on employee (first_name, last_name);

Output
index created.
sql> drop index employee_first_last_name_idx;
Output
index dropped.

5. Creating a Function-Based Index
Create table employee…….
create index employee_last_name_func_idx on employee(upper(last_name));
Output
index created.
sql> select first_name, last_name from employee where last_name = upper('price');
Output
no rows selected
sql>drop index employee_last_name_func_idx;
Output
index dropped
sql> -- clean the table
sql> drop table Employee;
Output
Table dropped.

6. Creates an index on the new added column
sql> alter table employee add employee_dup_id varchar2(7);
Output
table altered.
sql> update employee set employee_dup_id = employee_id;

Output
10 rows updated.
sql> create unique index employee_test_idx2 on employee(employee_dup_id);
Output
index created.
sql> drop table employee;
Output
table dropped.

7. Create index for upper case last name
create index upper_emp_idx on emp upper(lastname);
Output
Index created.

8. Create index along with the column definition
sql> create table emp2(emp_id number primary key using index
(create index pk_idx on emp2 (emp_id) tablespace users),
lastname varchar2(20) constraint lastname_create_nn not null,
firstname varchar2(15) constraint firstname_create_nn not null, phone varchar2(12),

company_name varchar2(50), constraint unique_emp_phone unique (phone) using index
(create index phone_idx on emp2 (phone) tablespace users));

Output

table created.

9. Create index for combined columns

create index fullname on emp(lastname, firstname);

Output

Index created.

10. Create unique index and check it in user_ind_columns and user_cons_columns

create unique index pk_idx on emp (emp_id);

sql> select index_name, table_name, column_name from user_ind_columns where table_name = 'E
MP';

sql> select constraint_name, table_name, column_name from user_cons_columns where table_name
= 'EMP';

SQL> drop table emp cascade constraints;

Output

Table dropped.

11. Create Non-Unique index

create table mytable (mytableid int primary key not null, name varchar(50),
phoneno varchar(15) default 'Unknown Phone');

Output

Table created.

sql> **create** index nameindex on mytable (name);

Output

Index created.

12. Modifying an Index

create table Employee……..

create index employee_last_name_idx on employee(upper(last_name));

Output

index created.

sql> select first_name, last_name from employee where last_name = upper('price');

Output

no rows selected

sql> alter index employee_last_name_idx rename to last_name_idx;

Output

index altered.

sql> drop index last_name_idx;

Output

index dropped.

sql> -- clean the table

sql> drop table employee;

Output

Table dropped.

4. SAVEPOINT

This command is used to temporarily save a transaction so that you can rollback to the point whenever necessary.

Create a table to execute the TCL commands.

Create table book(ISBN number, title varchar(25), author varchar(20), quantity number, price number, pub_year varchar(4));

Table created.

Insert the rows into table

Insert into book values(1001,'DBMS','Seema',321,215,2000);

1 row created.

Insert into book values(1002,'COA','John',350,400,2012);

1 row created.

Insert into book values(1003,'OS','Abraham',230,500,2001);

1 row created.

SQL>select*from book;

ISBN	TITLE	AUTHOR	QUANTITY	PRICE	PUB_YEAR
1001	DBMS	SEEMA	321	215	2000
1002	COA	JOHN	350	400	2012

1. Savepoint

Syntax

Savepoint savepoint-name;

Query

SQL>savepoint s1;

Savepoint created.

SQL>delete from books;

4 rows deleted.

SQL>select*from book;

No rows selected.

Syntax

Rollback to savepoint-name;

Query

SQL>rollback to s1;

Rollback complete.

SQL>select*from book;

ISBN	TITLE	AUTHOR	QUANTITY	PRICE	PUB_YEAR
1001	DBMS	SEEMA	321	215	2000
1002	COA	JOHN	350	400	2012

2. Release Savepoint

Release Savepoint command is used to remove a savepoint that you have created.

Syntax

Release savepoint savepoint_name;

Query

Release savepoint s1;

RESULT:

Thus SQL queries to retrieve information from the database using MySQL executed successfully.

Ex. No: 4
Date:

DATABASE PROGRAMMING:IMPLICIT AND EXPLICIT CURSORS

AIM

To create sql queries to implement implicit and explicit curses

CURSORS

A cursor is a pointer to this context area. PL/SQL controls the context area throughout a cursor. A cursor holds the rows(one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

There are 2 types of cursors

Implicit cursors
Explicit cursors

IMPLICIT CURSORS

They are created by oracle whenever an sql statement is executed, when there is no explicit cursor for that statement.

PROGRAM

```
Declare
  v_first_name varchar2(100);
  v_last_name varchar2(100);
begin
  select first_name,last_name
  into v_first_name,v_last_name
  from employees
  where employee_id=100;
  dbms_output.put_line(v_first_name|| ' '||v_last_name);
exception
  when NO_DATA_FOUND then
  dbms_output.put_line('No data found.Is the employee id valid?');
  when TOO_MANY_ROWS then
  dbms_output.put_line("Too many record." || "Are you identifying the employee        by id?"
end
/
```

EXPLICIT CURSORS

They are programmer defined cursors for gaining more control over the context area.

SYNTAX

CURSOR CURSOR_NAME IS SELECT STATEMENT;

PROGRAM

```
Declare
 CURSOR crs is SELECT * FROM employees;
 rec employee % rowtype;
begin
 if not crs% isopen then
  open crs;
```

```
 end if;
 loop
  fetch crs into rec;
  exit when crs% not found;
 dbms_output.put_line(rec.first_name|| ‘ ‘ || rec.last_name);
 end loop
 if crs% isopen then
  close crs;
 end if;
end
/
```

RESULT

Implementation of implicit and explicit cursors are done successfully

EX NO: 5 **DATE:**	**CREATION OF PROCEDURES**

AIM:

To create , study and implement PL/SQL procedures programs using MYSQL.

PROCEDURES:

Procedure is usually used to perform any specific task and functions which are used to compute a value . It is subprogram that performs specific action.

PARAMETERS:

1. In parameter : pass values to the subprogram when invoked,
2. Out parameter: return values to the caller of a subprogram
3. Inout parameter : pass initial values to subprogram when invoked and it also returns updated values to the caller.

SYNTAX:

```
Create[or replace] procedure procedurename
[(parameter_name [in|out|in out] type [,….])]
{is|as}
Begin
<procedure_body>
End procedure_name;
```

1. Display the string "Hello World"

```
SQL>set server output on
Create or replace procedure greetings
As
Begin
Dbms_output.put_line("Hello World");
End;
/
```

OUTPUT:

```
Hello World
Procedure created
```

2. Find the minimum of two values using IN and OUT parameter.

```
SQL>set serveroutput on
Declare
        A number;
        B number;
        C number;
        Procedure findmin(x in number, y in number , z out number) is
Begin   If x<y
        Z=x;
        Else
        Z=y;
        End if;

End;
Begin   A=23;
        B=45;
```

```
        Indmin(a,b,c);
        Dbms_output.put_line('minimum of (23,45):'||c);
End;
/
```

OUTPUT:

Minimum of (23,45) : 23
PL/SQL procedure successfully

3. Compute the square of value of a passed value using IN and OUT parameter:

```
SQL>set serveroutput on
Declare
        A number;
        Procedure squarenum(x in out number) is
Begin   X=x*x;

End;
Begin   A=23;
        Squarenum(a);
        Dbms_output.put_line('square of(23):'||a);

End;
/
```

OUTPUT:

Square of (23):529
PL/SQL procedure successfully completed.

CREATE TABLE:

Create table books(isn number,title varchar(11) , author varchar(11),quantity number,price number , year number);
Table created.
Create table orders(ordermo number,Cust_id number ,order_data date);
Table created.
Create table customer(cust_id number , cust_name varchar(10),address varchar(10),car_no number);
Table created
Create table order_list(orderno number,isbn number,quantity number,ship_data date);
Table created.

INSERT RECORD INTO THE TABLE:

insert into books values(1004, dbms', elmasri,321,237,2004),
1 row inserted.
insert into orders values(2500,1200'23/march/2006");
1 row inserted
insert into customer values(1200 ,'raja','chennai',8000);
1 row inserted.
insert into order list(2500,1006,3 ,'23/apr/2006');
1 row inserted

DISPLAY THE TABLE:

SQL>select * from books;

ISBN	TITLES	AUTHOR	QUANTITY	PRICE	PUB_YEAR
1004	DBMS	ELMASRI	321	237	2004
1006	PRINCIPLES	SEKAR	50	717	2006
1008	QUERIES	JOHN	43	345	2001
1005	HP	JKR	123	234	2006

SQL>select*from orders;

ORDERNO	CUST_ID	ORDER_DATA
2500	1200	23-MARCH-06
2600	1204	04-APR-06
2700	1202	20-DEC-06

SQL>select * from customer;

CUST_ID	CUST_NAME	ADDRESS	CARD_NO
1200	RAJA	CHENNAI	8000
1202	MITHUN	MUMBAI	8002
1204	VIJAY	JAIPUR	8004

SQL>select*from order_list;

ORDERNO	ISBN	QUANTITY	SHIP_DATA
2500	1006	3	23-APR-06
2600	1004	4	04-MAY-07
2700	1008	12	07-FEB-07

PROGRAM:

```
Create or replace procedure pro1(a1 in number a2 in number) is
begin
update books set quantity=quantity +a1 where books isbn=a2;
end up;
PROCEDURE CREATED.
SQL >begin
2 pro1(&quantity,&isbn);
3 end;
4/
Enter values for quantity :7
Enter the value for isbn:1005
old 2:pro1(&quantity,&isbn);
new 2:pro1(7,1005);
```

OUTPUT:

PL/SQL procedure successfully completed.

SOL>select * from books;

ISBN	TITLE	AUTHOR	QUANTITY	PRICE	PUB_YEAR
1004	DBMS	ELMASRI	321	237	2004

RESULT:

Thus PL/SQL procedures programs were implemented and executed successfully.

EX: NO. 6	CREATION OF DATABASE TRIGGERS
DATE:	

AIM:

. To create, study and implement PL/SQL triggers programs using MYSQL.

Triggers

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

Trigger statement: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.

Trigger body or trigger action: It is a PL/SQL block that is executed when the triggering statement is used.

Trigger restriction: Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

1. To generate data automatically
2. To enforce complex integrity constraints
3. To customize complex securing authorizations
4. To maintain the replicate table
5. To audit data modifications

Types of Triggers

The various types of triggers are as follows,

1. Before: It fires the trigger before executing the trigger statement.
2. After: It fires the trigger after executing the trigger statement.
3. For each row: It specifies that the trigger fires once per row.
4. For each statement: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

Variables Used in Triggers

1. :new
2. :old

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

Syntax:

Create or replace trigger <trg_name> Before /After Insert/Update/Delete
[of column_name, column_name….]

on <table_name>

[for each row]

[when condition]

Begin

---statement

end;

1. Create a trigger that insert current user into a username column of an existing table

Program

```
SQL> create table itstudent(name varchar2(15),username varchar2(15));
Table created.
SQL> create or replace trigger itstudent before insert on itstudent for each row
declare
 name varchar2(20);
 begin
 select user into name from dual;
 :new.username:=name;
 end;
 /
Trigger created.
```

Output:

```
SQL> insert into itstudent values('&name','&username');
Enter value for name: Ravi
Enter value for username: kumar
old 1: insert into itstudent values('&name','&username')
new 1: insert into itstudent values('Ravi','kumar')
1 row created.
SQL> /
Enter value for name: raj
Enter value for username: sundar
old 1: insert into itstudent values('&name','&username')
new 1: insert into itstudent values('raj','sundar')
1 row created.

SQL> select * from itstudent;
```

NAME	USERNAME
RAVI	SCOTT
RAJ	SCOTT

2. Before Insert Trigger

Create table

```
create table person (id int, name varchar(30), dob date, primary key(id));
```

Program

```
create or replace
```

```
trigger person_insert_before
before
insert
on person
for each row
begin
dbms_output.put_line('before insert of ' || :new.name);
end;
/
```

Output

```
insert into person(id,name,dob) values (1,'john doe','12/jan/1995');
1 row inserted
select * from person;
```

ID	NAME	DOB
1	JOHN DOE	12/JAN/1995

3. After Insert Trigger

Program

```
create or replace
trigger person_insert_after
after
insert
on person
for each row
begin
dbms_output.put_line('after insert of ' || :new.name);
end;
/
```

Output

```
insert into person(id, name, dob) values (2,'jane doe', '13/FEB/1994');
1 row inserted
select * from person;
```

ID	NAME	DOB
1	JOHN DOE	12/JAN/1995
2	JANE DOE	13/FEB/1995

4. Before Update Statement Trigger

Program

```
create or replace
trigger person_update_s_before
```

before update

on person

begin

dbms_output.put_line('before updating some person(s)');

end

Output

Update person set dob = sysdate;

2 rows updated.

select * from person;

ID	NAME	DOB
1	JOHN DOE	06/OCT/2014
2	JANE DOE	06/OCT/2014

5. Delete the ID from each row

Program

SQL> Create or replace trigger trig2

after delete on person for each row

begin

delete from person where id :old.id;

end;

/

Output:

SQL> delete person where id= 2

1 row deleted;

SQL> Select * from person;

ID	NAME	DOB
1	JOHN DOE	12/JAN/1995

Result

Thus PL/SQL Trigger was created and implemented successfully using MYSQL.

EXNO: 7

DATE:

WRITE A PL/SQL BLOCK THAT HANDLES ALL TYPES OF EXCEPTIONS

AIM:

Write a PL/SQL block that handles all types of exceptions.

EXCEPTIONS

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly.

Exceptions are error handling mechanisms. They are of 2 types,

1. Pre – defined exceptions
2. User – defined exceptions

To create the table 'ssitems' on which the exception handling Mechanisms are going to be performed

Create table

SQL> create table ssitems(id number(10), quantity number(10), actualprice number(10));

Table created.

Insert record into the table

SQL> insert into ssitems values(100,5,5000);

1 row created.

SQL> insert into ssitems values(101,6,9000);

1 row created.

SQL> insert into ssitems values(102,4,4000);

1 row created.

SQL> insert into ssitems values(103,2,2000);

1 row created.

SQL> select * from ssitems;

Display the table

ID	QUANTITY	ACTUALPRICE
100	5	5000
101	6	9000
102	4	4000
103	2	2000

Select * from ssitems

1. 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.

Syntax

begin

sequence of statements;

exception

when < exception name > then

sequence of statements;

end;

Program

```
SQL> set serveroutput on;
SQL> declare
price ssitems.actualprice % type;
begin
select actualprice into price from ssitems where quantity=10;
exception
when no_data_found then
dbms_output.put_line ('ssitems missing');
end;
/
```

Output

ssitems missing

PL/SQL procedure successfully completed.

Displaying the Updated Table

SQL> select * from ssitems;

ID	QUANTITY	ACTUALPRICE
100	5	5000
101	6	9000
102	4	4000
103	2	2000

2. 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.

Syntax

declare

< exception name > exception;

begin

sequence of statements;

raise < exception name >;

exception

when < exception name > then

sequence of statements;

end;

Program

```
SQL> set serveroutput on;
SQL> declare
zero_price exception;
price number(8,2);
begin
select actualprice into price from ssitems where id=103;
if price=0 or price is null then
raise zero_price;
end if;
exception
when zero_price then
dbms_output.put_line('Failed zero price');
end;
/
```

Output

PL/SQL procedure successfully completed.

Displaying the Updated Table

SQL> select * from ssitems;

ID	QUANTITY	ACTUALPRICE
100	5	5000
101	6	9000
102	4	4000
103	2	2000

Result

Thus PL/SQL block that handles all types of exceptions were implemented and executed successfully.

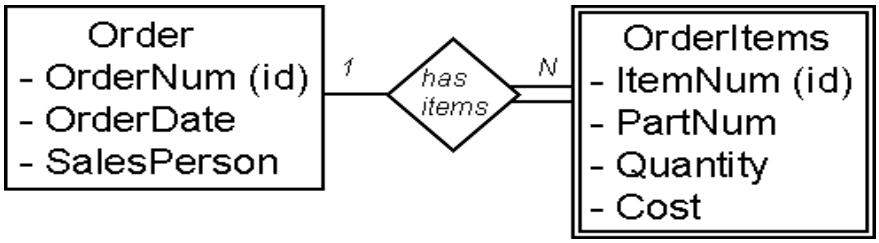
Ex. No: 8 Date:	DATABASE DESIGN USING E-R MODEL AND NORMALIZATION

AIM:

To design a simple database using E-R Model and Normalization techniques.

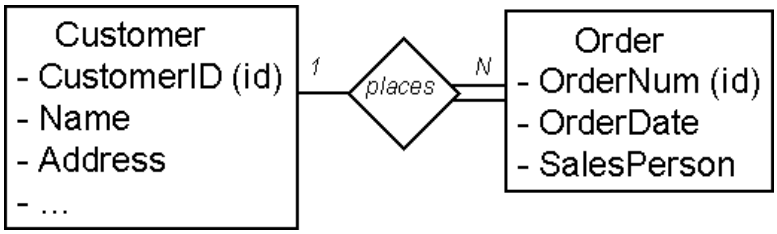
ER diagram:

Chen Notation



- **ORDER** (OrderNum (key), OrderDate, SalesPerson)
  ORDERITEMS (OrderNum (key)(fk) , ItemNum (key), PartNum, Quantity, Cost)
- In the above example, in the ORDERITEMS Relation: OrderNum is the *Foreign Key* and OrderNum plus ItemNum is the *Composite Key*.

Chen Notation



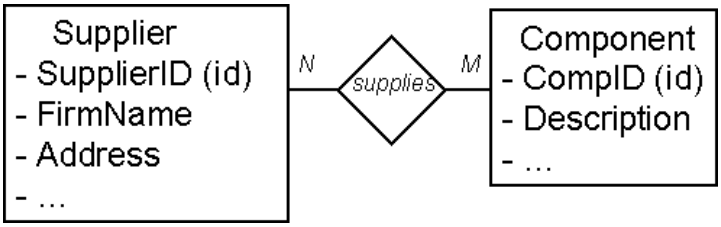
In the ORDER Relation: OrderNum is the *Key*.

Representing Relationships

- **1:1** Relationships. The key of one relation is stored in the second relation. Look at example queries to determine which key is queried most often.
- **1:N** Relationships.
- **Parent** - Relation on the "1" side.
- **Child** - Relation on the "Many" side.
- Represent each Entity as a relation.
  Copy the key of the parent into the child relation.
- **CUSTOMER (CustomerID (key), Name, Address, ...)**
  ORDER (OrderNum (key), OrderDate, SalesPerson, CustomerID (fk))
- **M:N** Relationships. Many to Many relationships cannot be directly implemented in relations.
- Solution: Introduce a third *Intersection relation* and copy keys from original two

relations.

Chen Notation



- SUPPLIER (SupplierID (key), FirmName, Address, ...) COMPONENT (CompID (key), Description, ...)
- SUPPLIER_COMPONENT (SupplierID (key), CompID (key))
- Note that this can also be shown in the ER diagram. Also, look for potential added attributes in the intersection relation.

RESULT:

Thus the ER Database design using E-R model and Normalization was implemented successfully.

Ex. No: 9 Date:	DESIGN AND IMPLEMENTATION OF BANKING SYSTEM

AIM:

To design and implement the Banking System using Visual Basic 6.0 as the front end and SQL plus as back end.

PROCEDURE:

1. Create the DB for banking system source request using SQL
2. Establishing ODBC connection
3. Click add button and select oracle in ORA home 90 click finished
4. A window will appear give the data source name as oracle and give the user id as scott 5. Now click the test connection a window will appear with server and user name give user as scott and password tiger Click ok
6. VISUAL BASIC APPLICATION:-
   - Create standard exe project in to and design ms from in request format
   - To add ADODC project select component and check ms ADO data control click ok
   - Now the control is added in the tool book
   - Create standard exe project in to and design ms from in request format
7. ADODC CONTEOL FOR ACCOUNT FROM:-

Click customs and property window and window will appear and select ODBC data source name as oracle and click apply as the some window.

CREATE A TABLE IN ORACLE

```
SQL>create table account(cname varchar(20),accno number(10),balance number);
Table Created

SQL> insert into account values('&cname',&accno,&balance);
Enter value for cname: Mathi
Enter value for accno: 1234
Enter value for balance: 10000
old 1: insert into account values('&cname',&accno,&balance) new
1: insert into emp values('Mathi',1234,10000)
1 row created.
```

SOURCE CODE FOR FORM1

```
Private Sub ACCOUNT_Click()
Form2.Show
End Sub
Private Sub
EXIT_Click()
Unload Me
```

```
End Sub
Private Sub
TRANSACTION_Click()
Form3.Show
End Sub
```

SOURCE CODE FOR FORM 2

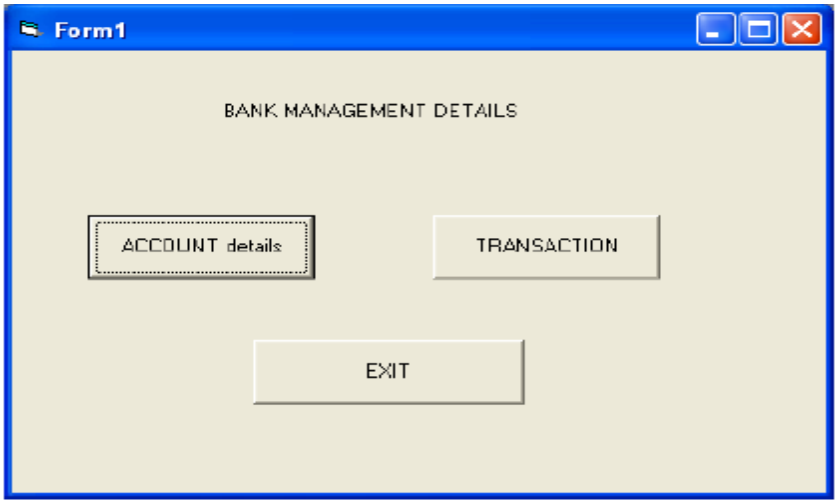
```
Private Sub CLEAR_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text    =    ""
End Sub
Private Sub
DELETE_Click()
Adodc1.Recordset.DELETE MsgBox "record deleted"
Adodc1.Recordset.MoveNext If Adodc1.Recordset.EOF = True Then
Adodc1.Recordset.MovePrevious
End If End
Sub
Private Sub EXIT_Click()
Unload Me
End Sub Private
Sub
HOME_Click()
Form1.Show End
Sub Private Sub
INSERT_Click() Adodc1.Recordset.AddNew
End Sub
Private Sub
TRANSACTION_Click()
Form3.Show
End Sub
Private Sub UPDATE_Click() Adodc1.Recordset.UPDATE MsgBox "record updated
successfully"
End Sub
```

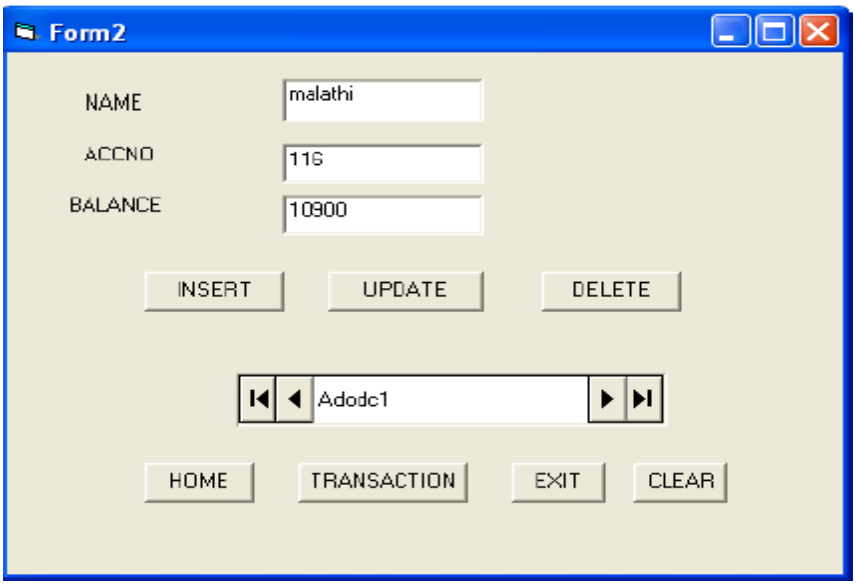
SOURCE CODE FOR FORM 3

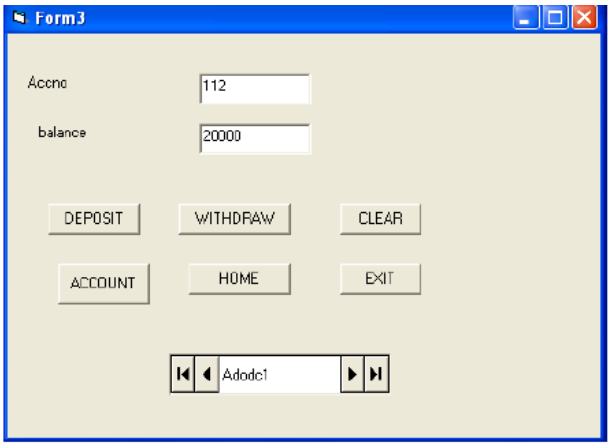
```
Private Sub ACCOUNT_Click()
Form2.Show
End Sub
Private Sub CLEAR_Click()
```

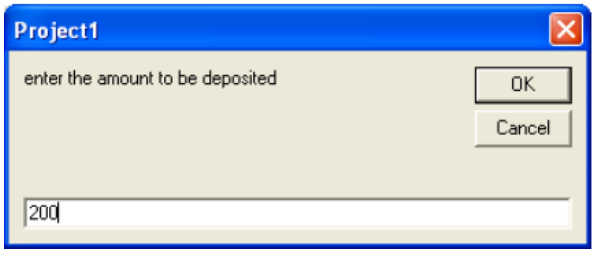
```
   Text1.Text = ""
Text2.Text =""
End Sub Private
Sub
   DEPOSIT_Click()

   Dim s As String s = InputBox("enter the amount to be deposited")
   Text2.Text = Val(Text2.Text) + Val(s) A = Text2.Text MsgBox "CURRENT BALANCE IS
   Rs" + Str(A) Adodc1.Recordset.Save Adodc1.Recordset.UPDATE
   End Sub Private
   Sub
   EXIT_Click()
    Unload Me End
   Sub Private Sub
   HOME_Click()
   Form1.Show End
   Sub Private Sub
   WITHDRAW_Click()
   Dim s As String s = InputBox("enter the amount to be deleted")
   Text2.Text = Val(Text2.Text) - Val(s) A = Text2.Text MsgBox "current balance is Rs" +
   Str(A)
Adodc1.Recordset.Save
Adodc1.Recordset.UPDATE
    End Sub
```

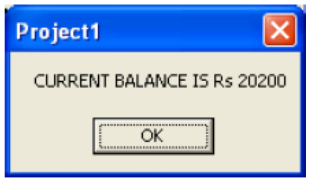
FORM DESIGN:


Form1
BANK MANAGEMENT DETAILS
ACCOUNT details
TRANSACTION
EXIT


Form2
NAME
malathi
ACCNO
116
BALANCE
10900
INSERT
UPDATE
DELETE
Adodc1
HOME
TRANSACTION
EXIT
CLEAR







RESULT:

Thus the banking system was designed and implemented successfully using Visual Basic and SQL Plus.

Ex. No: 10 Date:	AUTOMATIC BACKUP OF FILES AND RECOVERY

AIM:

To study about automatic backup of files and recovery.

INTRODUCTION:

Because data is the heart of the enterprise, it's crucial to protect it. And to protect organization's data, one need to implement a data backup and recovery plan. Backing up files can protect against accidental loss of user data, database corruption, hardware failures, and even natural disasters. It's our job as an administrator to make sure that backups are performed and that backup tapes are stored in a secure location.

Creating a Backup and Recovery Plan

Data backup is an insurance plan. Important files are accidentally deleted all the time. Mission-critical data can become corrupt. Natural disasters can leave office in ruin. With a solid backup and recovery plan, one can recover from any of these.

Figuring Out a Backup Plan

It takes time to create and implement a backup and recovery plan. We'll need to figure out what data needs to be backed up, how often the data should be backed up, and more. To help we create a plan, consider the following:

- How important is the data on systems? The importance of data can go a long way in helping to determine if one need to back it up—as well as when and how it should be backed up. For critical data, such as a database, one'll want to have redundant backup sets that extend back for several backup periods. For less important data, such as daily user files, we won't need such an elaborate backup plan, but'll need to back up the data regularly and ensure that the data can be recovered easily.
- What type of information does the data contain? Data that doesn't seem important to we may be very important to someone else. Thus, the type of information the data contains can help we determine if we need to back up the data—as well as when and how the data should be backed up.
- How often does the data change? The frequency of change can affect our decision on how often the data should be backed up. For example, data that changes daily should be backed up daily.
- How quickly do we need to recover the data? Time is an important factor in creating a backup plan. For critical systems, we may need to get back online swiftly. To do this, we may need to alter our backup plan.
- Do we have the equipment to perform backups? We must have backup hardware to perform backups. To perform timely backups, we may need several backup devices

  several sets of backup media. Backup hardware includes tape drives, optical drives, and removable disk drives. Generally, tape drives are less expensive but slower than other types of drives.

- Who will be responsible for the backup and recovery plan? Ideally, someone should be a primary contact for the organization's backup and recovery plan. This person may also be responsible for performing the actual backup and recovery of data.
- What is the best time to schedule backups? Scheduling backups when system use is as low as possible will speed the backup process. However, we can't always schedule backups for off-peak hours. So we'll need to carefully plan when key system data is backed up.
- Do we need to store backups off-site? Storing copies of backup tapes off-site is essential to recovering our systems in the case of a natural disaster. In our off-site storage location, we should also include copies of the software we may need to install to reestablish operational systems.

The Basic Types of Backup

There are many techniques for backing up files. The techniques use will depend on the type of data we're backing up, how convenient we want the recovery process to be, and more.

If we view the properties of a file or directory in Windows Explorer, we'll note an attribute called Archive. This attribute often is used to determine whether a file or directory should be backed up. If the attribute is on, the file or directory may need to be backed up. The basic types of backups we can perform include

- Normal/full backups all files that have been selected are backed up, regardless of the setting of the archive attribute. When a file is backed up, the archive attribute is cleared. If the file is later modified, this attribute is set, which indicates that the file needs to be backed up.
- Copy backups all files that have been selected are backed up, regardless of the setting of the archive attribute. Unlike a normal backup, the archive attribute on files isn't modified. This allows us to perform other types of backups on the files at a later date.
- Differential backups Designed to create backup copies of files that have changed since the last normal backup. The presence of the archive attribute indicates that the file has been modified and only files with this attribute are backed up. However, the archive attribute on files isn't modified. This allows performing other types of backups on the files at a later date.
- Incremental backups Designed to create backups of files that have changed since the most recent normal or incremental backup. The presence of the archive attribute indicates that the file has been modified and only files with this attribute are backed up. When a file is backed up, the archive attribute is cleared. If the file is later modified, this attribute is set, which indicates that the file needs to be backed up.
- Daily backups Designed to back up files using the modification date on the file itself. If a file has been modified on the same day as the backup, the file will be backed up. This technique doesn't change the archive attributes of files.

In we backup plan we'll probably want to perform full backups on a weekly basis and supplement this with daily, differential, or incremental backups. We may also want to create an extended backup set for monthly and quarterly backups that includes additional files that aren't

being backed up regularly.

Tip we'll often find that weeks or months can go by before anyone notices that a file or data source is missing. This doesn't mean the file isn't important. Although some types of data aren't used often, they're still needed. So don't forget that we may also want to create extra sets of backups for monthly or quarterly periods, or both, to ensure that we can recover historical data over time.

Differential and Incremental Backups

The difference between differential and incremental backups is extremely important. To understand the distinction between them. As it shows, with differential backups we back up all the files that have changed since the last full backup (which means that the size of the differential backup grows over time). With incremental backups, we only back up files that have changed since the most recent full or incremental backup (which means the size of the incremental backup is usually much smaller than a full backup).

Once we determine what data we're going to back up and how often, we can select backup devices and media that support these choices. These are covered in the next section.

Selecting Backup Devices and Media

Many tools are available for backing up data. Some are fast and expensive. Others are slow but very reliable. The backup solution that's right for our organization depends on many factors, including

- Capacity The amount of data that we need to back up on a routine basis. Can the backup hardware support the required load given our time and resource constraints?
- Reliability The reliability of the backup hardware and media. Can we afford to sacrifice reliability to meet budget or time needs?
- Extensibility The extensibility of the backup solution. Will this solution meet our needs as the organization grows?
- Speed the speed with which data can be backed up and recovered. Can we afford to sacrifice speed to reduce costs?
- Cost the cost of the backup solution. Does it fit into our budget?

Common Backup Solutions

Capacity, reliability, extensibility, speed, and cost are the issues driving our backup plan. If we understand how these issues affect our organization, we'll be on track to select an appropriate backup solution. Some of the most commonly used backup solutions include

- Tape drives Tape drives are the most common backup devices. Tape drives use magnetic tape cartridges to store data. Magnetic tapes are relatively inexpensive but aren't highly reliable. Tapes can break or stretch. They can also lose information over time. The average capacity of tape cartridges ranges from 100 MB to 2 GB. Compared with other

backup solutions, tape drives are fairly slow. Still, the selling point is the low cost.

- Digital audio tape (DAT) drives DAT drives are quickly replacing standard tape drives as the preferred backup devices. DAT drives use 4 mm and 8 mm tapes to store data. DAT drives and tapes are more expensive than standard tape drives and tapes, but they offer more speed and capacity. DAT drives that use 4 mm tapes can typically record over 30 MB per minute and have capacities of up to 16 GB. DAT drives that use 8 mm tapes can typically record more than 10 MB per minute and have capacities of up to 36 GB (with compression).
- Auto-loader tape systems Auto-loader tape systems use a magazine of tapes to create extended backup volumes capable of meeting the high-capacity needs of the enterprise. With an auto-loader system, tapes within the magazine are automatically changed as needed during the backup or recovery process. Most auto-loader tape systems use DAT tapes. The typical system uses magazines with between 4 and 12 tapes. The main drawback to these systems is the high cost.
- Magnetic optical drives Magnetic optical drives combine magnetic tape technology with optical lasers to create a more reliable backup solution than DAT. Magnetic optical drives use 3.5-inch and 5.25-inch disks that look similar to floppies but are much thicker. Typically, magnetic optical disks have capacities of between 1 GB and 4 GB.
- Tape jukeboxes Tape jukeboxes are similar to auto-loader tape systems. Jukeboxes use magnetic optical disks rather than DAT tapes to offer high-capacity solutions. These systems load and unload disks stored internally for backup and recovery operations. Their key drawback is the high cost.
- Removable disks Removable disks, such as Iomega Jaz, are increasingly being used as backup devices. Removable disks offer good speed and ease of use for a single drive or single system backup. However, the disk drives and the removable disks tend to be more expensive than standard tape or DAT drive solutions.
- Disk drives Disk drives provide the fastest way to back up and restore files. With disk drives, you can often accomplish in minutes what takes a tape drive hours. So when business needs mandate a speedy recovery, nothing beats a disk drive. The drawbacks to disk drives, however, are relatively high costs and less extensibility.

Before we can use a backup device, we must install it. When we install backup devices other than standard tape and DAT drives, we need to tell the operating system about the controller card and drivers that the backup device uses. For detailed information on installing devices and drivers, see the section of Chapter 2 entitled "Managing Hardware Devices and Drivers."

Buying and Using Tapes

Selecting a backup device is an important step toward implementing a backup and recovery plan. But we also need to purchase the tapes or disks, or both, that will allow we to implement our plan. The number of tapes we need depends on how much data we'll be backing up, how often we'll be backing up the data, and how long we'll need to keep additional data sets.

The typical way to use backup tapes is to set up a rotation schedule whereby we rotate through

two or more sets of tapes. The idea is that we can increase tape longevity by reducing tape usage and at the same time reduce the number of tapes we need to ensure that we have historic data on hand when necessary.

One of the most common tape rotation schedules is the 10-tape rotation. With this rotation schedule, we use 10 tapes divided into two sets of 5 (one for each weekday). As shown in Table 14-2, the first set of tapes is used one week and the second set of tapes is used the next week. On Fridays, full backups are scheduled. On Mondays through Thursdays, incremental backups are scheduled. If we add a third set of tapes, we can rotate one of the tape sets to an off-site storage location on a weekly basis.

Tip The 10-tape rotation schedule is designed for the 9 to 5 workers of the world. If we're in a 24 x 7 environment, we'll definitely want extra tapes for Saturday and Sunday. In this case, use a 14-tape rotation with two sets of 7 tapes. On Sundays, schedule full backups. On Mondays through Saturdays, schedule incremental backups.

RESULT:

Thus the study of automatic backup of files was studied successfully.