



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Estd : 2008

Accredited by NAAC with A+ and NBA

Affiliated to Osmania University & Approved by AICTE



---

## VISION

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

## MISSION

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice.

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices.

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Affiliated to Osmania University & Approved by AICTE



DEPARTMENT  
OF  
COMPUTER SCIENCE AND ENGINEERING

# **DATABASE MANAGEMENT SYSTEM LAB**

**Prepared**  
**By**  
**B VASAVI SRAVANTHI,**  
**Assistant Professor,**  
**Department of CSE.**



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Affiliated to Osmania University & Approved by AICTE



Estd : 2008

## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

### **VISION & MISSION**

#### **VISION**

To become a leader in providing Computer Science & Engineering education with emphasis on knowledge and innovation.

#### **MISSION**

- To offer flexible programs of study with collaborations to suit industry needs.
- To provide quality education and training through novel pedagogical practices.
- To expedite high performance of excellence in teaching, research and innovations.
- To impart moral, ethical values and education with social responsibility.



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)



Estd : 2008

Accredited by NAAC with A+ and NBA

Affiliated to Osmania University & Approved by AICTE

## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

### **PROGRAM EDUCATIONAL OBJECTIVES**

After 3-5 years of graduation, the graduates will be able to

**PEO1:** Apply technical concepts, Analyze, synthesize data to Design and create novel products and solutions for the real life problems.

**PEO2:** Apply the knowledge of Computer Science Engineering to pursue higher education with due consideration to environment and society.

**PEO3:** Promote collaborative learning and spirit of team work through multidisciplinary projects

**PEO4:** Engage in life-long learning and develop entrepreneurial skills.



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Estd : 2008

Affiliated to Osmania University & Approved by AICTE



## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### PROGRAM OUTCOMES

**Engineering Graduates will be able to:**

**P01. Engineering knowledge:** Apply the basic knowledge of mathematics, science and engineering fundamentals along with the specialized knowledge of mechanical engineering to understand complex engineering problems.

**P02. Problem analysis:** Identify, formulate, design and analyze complex mechanical engineering problems using knowledge of science and engineering.

**P03. Design/development of solutions:** Develop solutions for complex engineering problems, design and develop system components or processes that meet the specified needs with appropriate consideration of the public health and safety, and the cultural, societal, and environmental considerations.

**P04. Conduct investigations of complex problems:** Formulate engineering problems, conduct investigations and solve using research-based knowledge.

**P05. Modern tool usage:** Use the modern engineering skills, techniques and tools that include IT tools necessary for mechanical engineering practice.

**P06. The engineer and society:** Apply the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**P07. 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.

**P08. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities during professional practice.

**P09. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)



Accredited by NAAC with A+ and NBA

Estd : 2008

Affiliated to Osmania University & Approved by AICTE

**P010. Communication:** Communicate effectively on complex engineering activities to various groups, ability to write effective reports and make effective presentations.

**P011. Project management and finance:** Demonstrate and apply the knowledge to understand the management principles and financial aspects in multidisciplinary environments.

**P012. 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**

**At the end of 4 years, Computer Science and Engineering graduates at MCET will be able to:**

**PSO1:** Apply the knowledge of Computer Science and Engineering in various **domains** like networking and data mining to manage projects in multidisciplinary environments.

**PSO2:** Develop software applications with open-ended programming environments.

**PSO3:** Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE DESCRIPTION FILE**

Academic Year & Semester	2022-2023, Semester III
Course Code	1PC351AD
Course Title	DATA BASE MANAGEMENT SYSTEMS Lab
Curriculum Regulation	AUTONOMOUS
Semester	III
Course Instructor	Mrs B VASAVI SRAVANTHI, Assistant Professor, CSE Department

**I. PREREQUISITE(S):**

Level	Credits	Semester	Prerequisites
UG	1	III	Prior programming experience with C, and basic mathematical knowledge is required for this course.

**II. SCHEME OF INSTRUCTIONS**

Lectures	Tutorials	Practicals	Credits
-	-	2	1

**III. SCHEME OF EVALUATION & GRADING**

S. No	Component	Duration	Maximum Marks
	Continuous Internal Evaluation (CIE)	2hrs	40
	CIE (Total)		40
5.	Semester End Practical Examination (University Examination)	3 hours	60
		<b>TOTAL</b>	<b>100</b>

Marks Range	85-100	70 to < 85	60 to < 70	55 to < 60	50 to < 55	40 to < 50	< 40	Absent
Grade	S	A	B	C	D	E	F	Ab
Grade Point	10	9	8	7	6	5	0	-



## DATABASE MANAGEMENT SYSTEMS LAB

**Semester III** **L** **T** **P** **Credits**

**Subject code – IPC351AD** **0** **0** **2** **1**

**Prerequisites: C Language**

Course Objectives:	Course Outcomes:
<ul style="list-style-type: none"> <li>➤ To practice various DDL, DML commands in SQL</li> <li>➤ To write simple and Complex queries in SQL</li> <li>➤ To practice various Functions, Joins &amp; sub queries in SQL</li> <li>➤ To write PL/SQL using cursors and collections</li> <li>➤ To write PL/SQL using Stored Procedures</li> </ul>	<ol style="list-style-type: none"> <li>1. Design and implement a database schema for a given problem</li> <li>2. Develop the query statements with the help of structured query language.</li> <li>3. Populate and query a database using SQL and PL/SQL</li> <li>4. Develop multi-user database application</li> <li>5. Design and implement E-R model for the given requirements</li> </ol>

### List of Programs:

1. Creation of database Tables (exercising the all SQL commands)
2. Simple and complex condition query creation using SQL Plus
3. Creation of database Tables using Integrity constraints and Functions
4. Simple and complex condition query creation using Joins
5. Simple and complex condition query creation using Sub queries and set operators
6. Creation of Views (exercising the all types of views)
7. Writing PL/SQL function and cursors
8. Writing PL/SQL stored procedure and triggers
9. Creation of Forms and reports for student Information, library information, Pay roll etc.
10. Case Study: Design Database for Bank
  - => Collect the information Related with Bank organization
  - => Draw E-R Diagrams for Bank
  - => Reduce E-R Diagrams to tables
  - => Normalize your Database up to 3<sup>rd</sup> Normal form
  - => Retrieve Bank information using SQL commands





**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Estd : 2008

Accredited by NAAC with A+ and NBA

Affiliated to Osmania University & Approved by AICTE



## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Outcomes (CO's):

**SUBJECT NAME: DATA BASE MANAGEMENT SYSTEM LAB CODE: 1PC351AD**

**SEMESTER: III**

CO No.	Course Outcomes	Taxonomy Level
1PC351AD.1	Design and implement a database schema for a given problem	Understanding
1PC351AD.2	Develop the query statements with the help of structured query language.	Apply
1PC351AD.3	Populate and query a database using SQL and PL/SQL	Apply
1PC351AD.4	Develop multi-user database application	Apply
1PC351AD.5	Design and implement E-R model for the given requirements	Understanding



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Estd : 2008

Affiliated to Osmania University & Approved by AICTE



## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

### **GENERAL LABORATORY INSTRUCTIONS**

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
  - a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
  - b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
  - c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviours with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out. If anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system/seat is kept properly.

**Head of the Department**

**Principal**



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Estd : 2008

Accredited by NAAC with A+ and NBA  
Affiliated to Osmania University & Approved by AICTE



## **CODE OF CONDUCT FOR THE LABORATORY**

- All students must observe the dress code while in the laboratory
- Footwear is NOT allowed
- Foods, drinks and smoking are NOT allowed
- All bags must be left at the indicated place
- The lab timetable must be strictly followed
- Be PUNCTUAL for your laboratory session
- All programs must be completed within the given time
- Noise must be kept to a minimum
- Workspace must be kept clean and tidy at all time
- All students are liable for any damage to system due to their own negligence
- Students are strictly PROHIBITED from taking out any items from the laboratory
- Report immediately to the lab programmer if any damages to equipment

## **BEFORE LEAVING LAB:**

- Arrange all the equipment and chairs properly.
- Turn off / shut down the systems before leaving.
- Please check the laboratory notice board regularly for updates.



## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### LIST OF EXPERIMENTS

Sl.No.	Name of the Experiment	Date of Experiment	Date of Submission	Page No	Faculty Signature
1.	Creation of database Tables (exercising the all SQL commands)	31/10/22	7/11/22		
2	Simple and complex condition query creation using SQL Plus	7/11/22	14/11/22		
3	Creation of database Tables using Integrity constraints and Functions	14/11/22	28/11/22		
4	Simple and complex condition query creation using Joins	28/11/22	5/12/22		
5.	Simple and complex condition query creation using Sub queries and set operators	28/11/22	5/12/22		
6.	Creation of Views (exercising the all types of views)	5/12/22	12/12/22		
7.	Writing PL/SQL function and cursors	5/12/22 & 12/12/22	12/12/22		
8	Writing PL/SQL stored procedure and triggers	12/12/22	19/12/22		
9	Creation of Forms and reports for student Information, library information, Pay roll etc.	19/12/22	2/1/23		



10	Case Study: Design Database for Bank => Collect the information Related with Bank organization => Draw E-R Diagrams for Bank => Reduce E-R Diagrams to tables => Normalize your Database up to 3 <sup>rd</sup> Normal form => Retrieve Bank information using SQL commands	2/1/23	21/1/23		
----	---	--------	---------	--	--



## **INTRODUCTION**

### **RDBMS:**

RDBMS stands for Relational Database Management System. RDBMS data is structured in database tables, fields and records. Each RDBMS table consists of database table rows. Each database table row consists of one or more database table fields. RDBMS store the data into collection of tables, which might be related by common fields (database table columns). RDBMS also provide relational operators to manipulate the data stored into the database tables. Most RDBMS use MYSQL as database query language. The most popular RDBMS are MS MYSQL Server, DB2, Oracle and MYSQL.

The relational model is an example of record-based model. Record based models are so named because the database is structured in fixed format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record types. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model. The relational model was designed by the IBM research scientist and mathematician, Dr. E.F.Codd. Many modern DBMS do not conform to the Codd's definition of a RDBMS, but nonetheless they are still considered to be RDBMS. Two of Dr. Codd's main focal points when designing the relational model were to further reduce data redundancy and to improve data integrity within database systems.

The relation is the only data structure used in the relational data model to represent both entities and relationships between them. Rows of the relation are referred to as tuples of the relation and columns are its attributes. Each attribute of the column is drawn from the set of values known as domain

### **SQL**

SQL stands for Structured Query Language. Structured Query Language (MYSQL) which uses a combination of Relational algebra and Relational calculus. MYSQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. It is a data sub language used to organize, manage and retrieve data from relational database, which is managed by Relational Database Management System (RDBMS).

### **MYSQL:**

MYSQL is a database management system that allows you to manage relational databases. It is open-source software backed by Oracle. It means you can use MYSQL without paying a dime. Also, if you want, you can change its source code to suit your needs.

Even though MYSQL is open-source software, you can buy a commercial license version from Oracle to get a premium support service. MYSQL is pretty easy to master in comparison with other database software like Oracle Database, or Microsoft MYSQL Server. MYSQL can run on various platforms UNIX, Linux, Windows, etc. You can install it on a server or even in a desktop. Besides, MYSQL is reliable, scalable, and fast. If you develop websites or web applications, MYSQL is a good choice. MYSQL is an essential component of LAMP stack, which includes Linux, Apache, MYSQL, and PHP.





## DATA TYPES IN MYSQL

Data types define the nature of the data that can be stored in a particular column of a table

### Numeric Data types

Numeric data types are used to store numeric values. It is very important to make sure range of your data is between lower and upper boundaries of numeric data types.

TINYINT()	-128 to 127 normal      0 to 255 UNSIGNED.
SMALLINT()	-32768 to 32767 normal      0 to 65535 UNSIGNED.
MEDIUMINT()	-8388608 to 8388607 normal      0 to 16777215 UNSIGNED.
INT()	-2147483648 to 2147483647 normal      0 to 4294967295 UNSIGNED.
BIGINT()	-9223372036854775808 to 9223372036854775807 normal 0 to 18446744073709551615 UNSIGNED.
FLOAT	A small approximate number with a floating decimal point.
DOUBLE( , )	A large number with a floating decimal point.
DECIMAL( , )	A DOUBLE stored as a string, allowing for a fixed decimal point. Choice for storing currency values.

### Text Data Types

As data type category name implies these are used to store text values. Always make sure your length of your textual data do not exceed maximum lengths.

CHAR()	A fixed section from 0 to 255 characters long.
VARCHAR( )	A variable section from 0 to 255 characters long.
TINYTEXT	A string with a maximum length of 255 characters.
TEXT	A string with a maximum length of 65535 characters.
BLOB	A string with a maximum length of 65535 characters.
MEDIUMTEXT	A string with a maximum length of 16777215 characters.
MEDIUMBLOB	A string with a maximum length of 16777215 characters.
LONGTEXT	A string with a maximum length of 4294967295 characters.
LOB	A string with a maximum length of 4294967295 characters.

### Date / Time

DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS

Apart from above there are some other data types in MySQL.

ENUM	To store text value chosen from a list of predefined text values
SET	This is also used for storing text values chosen from a list of predefined text values. It can have multiple values.
BOOL	Synonym for TINYINT(1), used to store Boolean values
BINARY	Similar to CHAR, difference is texts are stored in binary format.
VARBINARY	Similar to VARCHAR, difference is texts are stored in binary format.

**Standard MYSQL** commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

**MYSQL** commands are divided into categories like DML (Data Manipulation language), DDL (Data definition language), TCL (Transaction control language) and DCL (Data control language). Here are a list of MYSQL commands.



## 1. DDL COMMANDS

CREATE , ALTER,DROP

## 2. DML COMMANDS

INSERT, UPDATE, SELECT, DELETE

## 3. TCL COMMANDS

COMMIT, ROLLBACK

## 4. DCL COMMANDS

GRANT, REVOKE

## 1. DDL COMMANDS

### CREATE TABLE

This statement is used to create a table. The syntax for this command is  
create table tablename (colname1 datatype [constraint], colname2 datatype [constraint]);

### ALTER TABLE

This command is used to add, drop columns in a table. The syntax for this command is  
alter table tablename add colname1

datatype [constraint];

alter table tablename drop column  
colname1;

### DROP TABLE

The syntax for this command is-  
drop table tablename;

## 2. DML COMMANDS

### INSERT ROWS

The syntax for this command is

insert into tablename(colname1,colname2) values(value1,value2);

### UPDATE ROWS

The syntax for this command is

update tablename set colname1=colvalue where colname2=colvalue;

### SELECT ROWS

This command is used to select rows from a table.

The syntax for this command is

select colname1,colname2 from tablename;

### DELETE ROWS

The syntax for this command is delete from tablename where [search\_conditions];

## 3. TCL COMMANDS

### COMMIT

This command is used for save the work done.

The syntax is:

COMMIT;

### ROLLBACK

This command is used to restore the database to original since the last commit.

The syntax is-

ROLLBACK;

## 4. DCL COMMANDS



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)



**Accredited by NAAC with A+ and NBA**

Estd : 2008

**Affiliated to Osmania University & Approved by AICTE**

## **GRANT**

This command is used for gives access privileges to users for database. The syntax is-  
GRANT dba to username;

## **REVOKE**

This command is used for withdraws access privileges to users for database.

The syntax is-

REVOKE permissions on table name from username;



## EXPERIMENT 1:

### AIM: Exercising the DDL, DML Commands

#### Basic MYSQL DDL Commands.

To practice basic MYSQL DDL Commands such as CREATE, ALTER, DROP, etc.

#### a. MYSQL - CREATE TABLE

CREATE TABLE Emp ( EmpNo number, EName VarChar(15), Job Char(10), Hiredate Date, Sal number(7,2), Comm. Number(7,2), Deptno number(3), Age number(3));

RESULT: Table created

#### b. MYMYSQL - ALTER TABLE

INPUT:

ALTER TABLE EMP ADD CONSTRAINT Pkey1 PRIMARY KEY (EmpNo);

RESULT: Table Altered

#### c. MYMYSQL - DROP TABLE

– Deletes table structure – Cannot be recovered – Use with caution

INPUT:

DROP TABLE EMP; Here EMP is table name

RESULT: Table Dropped.

Get the description of EMP table.

**desc emp;**

RESULT:

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		CHAR (10)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(3)
AGE		NUMBER(3)

#### Basic MYMYSQL DML Commands.

To practice basic MYMYSQL DML Commands such as INSERT, SELECT, UPDATE DELETE, etc.

#### a. MYMYSQL - INSERT INTO (Inserting multiple rows)



```
INSERT ALL INTO EMP(EmpNo,ENAME,Job,HireDate,Sal,Comm,DeptNo,Age)values (7369,'
SMITH',' CLERK','17-DEC-80',800,0,20,25)
INTO EMP(EmpNo,ENAME,Job,HireDate,Sal,Comm,DeptNo,Age)
values(7499,'ALLEN','SALESMAN','20-FEB-81',1600,300,30,25)
INTO EMP(EmpNo,ENAME,Job,HireDate,Sal,Comm,DeptNo,Age)
values (7521,' WARD','SALESMAN','22-FEB-81',1250,500,30,25)
INTO EMP(EmpNo,ENAME,Job,HireDate,Sal,Comm,DeptNo,Age)
values(7566,'JONES','MANAGER','02-APR-81',2975,500,20,25)
INTO EMP(EmpNo,ENAME,Job,HireDate,Sal,Comm,DeptNo,Age)
values(7698,'BLAKE','MANAGER','01-MAY-81',2850,1400,30,25)
select * from dual
```

## b. MYMYSQL - UPDATE

Examples:

UPDATE EMP SET Sal=20000 where EmpNo=7369;

RESULT:

1 row updated.

## c. MYMYSQL - SELECT

List all employee details.

select \* from emp;

RESULT:

EMPNO	ENAME	JOB	HIREDATE	SAL	COMM	DEPTNO	AGE
7369	SMITH	CLERK	17-DEC-80	800	0	20	25
7499	ALLEN	SALESMAN	20-FEB-81	1600	300	30	25
7521	WARD	SALESMAN	22-FEB-81	1250	500	30	25
7566	JONES	MANAGER	02-APR-81	2975	500	20	25
7698	BLAKE	MANAGER	01-MAY-81	2850	1400	30	25

## d. MYMYSQL - DELETE

Examples:

INPUT:

Delete from emp where empno=7369;

RESULT:

1 row deleted

List all employee details.

select \* from emp;

RESULT:

EMPNO	ENAME	JOB	HIREDATE	SAL	COMM	DEPTNO	AGE
7499	ALLEN	SALESMAN	20-FEB-81	1600	300	30	25
7521	WARD	SALESMAN	22-FEB-81	1250	500	30	25
7566	JONES	MANAGER	02-APR-81	2975	500	20	25
7698	BLAKE	MANAGER	01-MAY-81	2850	1400	30	25



**Basic MYMYSQL DCL Commands.**

To practice basic MYMYSQL DCL Commands such as GRANT, REVOKE, etc.

Syntax:

MYMYSQL>CREATING A USER

MYSQL>Connect system/manager;

MYSQL>Create user "username"identified by "password"

MYSQL>Grant DBA to "username"

MYSQL>Connect "username"/" password";

Example:

MYSQL>Connect 160615733052/123;

MYSQL>Create user cse2 identified by "cse"

MYSQL>Grant DBA to cse2

MYSQL>Connect cse2/cse;

RESULT:

Connected

Syntax

MYSQL>Grant privileges on object\_name to user-name;

Example:

MYSQL>Grant select,update on emp to cse2;

RESULT:

Grant succeeded

Syntax:

MYSQL>Revoke privileges on object\_name from user-name;

Example:

Revoke select,update on emp from cse2;

RESULT:

Revoke succeeded

**Basic MYSQL TCL Commands.**

To practice basic MYSQL TCL Commands such as COMMIT, SAVEPOINT, ROLLBACK.

Syntax:

MYSQL>Commit;

Example:

MYSQL>Commit;

Result:

Commit completed

Syntax:

MYSQL>Savepoint ID

Example:

MYSQL>Savepoint xyz;

Result:

Savepoint completed





Syntax:

MYSQL>Rollback to SavepointID

Example:

MYSQL>Rollback to xyz;

Result:

Rollback completed

## AIM : SIMPLE QUERIES AND COMPLEX QUERIES

### LIKE operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

#### The LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

a. MYSQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

List all employees which starts with either J or T.

INPUT

select ename from emp where ename like 'J%' or ename like 'A%';

RESULT:

ENAME

-----

JONES

ALLEN

### LOGICAL OPERATORS:

#### AND, OR, NOT

"OR" Logical Operator:

If we want to select rows that satisfy at least one of the given conditions, you can use the logical operator, OR.

**For example:** if you want to find the names of students who are studying either Maths or Science, the query would be like,

```
SELECT first name, last name, subject FROM student details WHERE subject = 'Maths' OR
subject = 'Science'
```

The output would be something like,

<u>first name</u>	<u>last name</u>	<u>subject</u>
-------------------	------------------	----------------

Anajali	Bhagwat	Maths
---------	---------	-------

Shekar	Gowda	Maths
--------	-------	-------

Rahul	Sharma	Science
-------	--------	---------

Stephen	Fleming	Science
---------	---------	---------

"AND" Logical Operator:

If you want to select rows that must satisfy all the given conditions, you can use the logical operator, AND.

**For Example:** To find the names of the students between the age 10 to 15 years, the query would be like:



SELECT first\_name, last\_name, age FROM student\_details WHERE age >= 10 AND age <= 15;  
The output would be something like,

first\_name   last\_name   age

Rahul	Sharma	10
Anajali	Bhagwat	12
Shekar	Gowda	15

"NOT" Logical Operator:

If you want to find rows that do not satisfy a condition, you can use the logical operator, NOT. NOT results in the reverse of a condition. That is, if a condition is satisfied, then the row is not returned.

**For example:** If you want to find out the names of the students who do not play football, the query would be like:

SELECT first\_name, last\_name, games FROM student\_details WHERE NOT games = 'Football'

The output would be something like,

first\_name   last\_name   Games

Rahul	Sharma	Cricket
Stephen	Fleming	Cricket
Shekar	Gowda	Badminton
Priya	Chandra	Chess

### Set Operators

Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries. MYSQL set operators.

Operator	Returns
UNION	All rows selected by either query.
UNION ALL	All rows selected by either query, including all duplicates.
INTERSECT	All distinct rows selected by both queries.
MINUS	All distinct rows selected by the first query but not the second.

All set operators have equal precedence. If a MYSQL statement contains multiple set operators, Oracle evaluates them from the left to right if no parentheses explicitly specify another order. The corresponding expressions in the select lists of the component queries of a compound query must match in number and datatype. If component queries select character data, the datatype of the return values are determined as follows:

- If both queries select values of datatype CHAR, the returned values have datatype CHAR
- If either or both of the queries select values of datatype VARCHAR2, the returned values have datatype VARCHAR2.

### Examples

Consider these two queries and their results:

SELECT part FROM orders\_list1;

PART



SPARKPLUG  
FUEL PUMP  
FUEL PUMP  
TAILPIPE

SELECT part FROM orders\_list2;

PART

CRANKSHAFT  
TAILPIPE  
TAILPIPE

The following examples combine the two query results with each of the set operators.

UNION Example

The following statement combines the results with the UNION operator, which eliminates duplicate

selected rows. This statement shows how datatype must match when columns do not exist in one or the other table:

SELECT part, partnum, to\_date(null) date\_in FROM orders\_list1

UNION

SELECT part, to\_date(null), date\_in FROM orders\_list2;

PART PARTNUM DATE\_IN

SPARKPLUG 3323165 10/24/98  
FUEL PUMP 3323162 12/24/99  
TAILPIPE 1332999 01/01/01  
CRANKSHAFT 9394991 09/12/02

SELECT part FROM orders\_list1 UNIONSELECT part FROM orders\_list2;

PART

SPARKPLUG  
FUEL PUMP  
TAILPIPE  
CRANKSHAFT

INTERSECT Example

The following statement combines the results with the INTERSECT operator, which returns only those rows returned by both queries:

SELECT part FROM orders\_list1

INTERSECT

SELECT part FROM orders\_list2;

PART

TAILPIPE



### MINUS Example

The following statement combines results with the MINUS operator, which returns only rows returned by the first query but not by the second:

```
SELECT part FROM orders_list1
MINUS
SELECT part FROM orders_list2;
PART
```

```
-----
SPARKPLUG
FUEL PUMP
```

### MYSQL GROUP BY Statement

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

#### MYSQL GROUP BY Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
```

#### MYSQL GROUP BY Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the total sum (total order) of each customer.

We will have to use the GROUP BY statement to group the customers.

We use the following MYSQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders GROUP BY Customer
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Hansen	2000
Nilsen	1700
Jensen	2000

### MYSQL HAVING Clause

The HAVING clause was added to MYSQL because the WHERE keyword could not be used with aggregate functions.

#### MYSQL HAVING Syntax

```
SELECT column_name, aggregate_function(column_name) FROM table_name WHERE
column_name operator value GROUP BY column_name HAVING
aggregate_function(column_name) operator value
```



### MYSQL HAVING Example

Now we want to find if any of the customers have a total order of less than 2000.

We use the following MYSQL statement:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Nilsen	1700

### MYSQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

MYSQL ORDER BY Syntax

```
SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC
```

ORDER BY Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name .We use the following SELECT statement:

```
SELECT * FROM Persons ORDER BY LastName
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

### ORDER BY DESC Example

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.

We use the following SELECT statement:

```
SELECT * FROM Persons
ORDER BY LastName DESC
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes



3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

Complex condition query creation

## NESTED QUERIES

Create a tables studetail,stumarks and stucourse

Studetail(sno,sname,sphno) sno-primary key

Stumarks(sno,smarks) sno-foreign key of studetail(sno)

Stucourse(sno,scourse) sno-foreign key of studetail(sno)

## QUERIES

1.Display student details who scored 70 in DBMS

Select \* from student where sno in (select sno from stumarks where smarks=70 and sno in (select sno from stucourse where scourse='DBMS'))

OUTPUT:

SNO	SNAME	SPHNO
101	ALEKHIA	228601

2. Display student who enrolled for CN

Select sname from student wheresno in (select sno from stucourse where scourse='CN')

OUTPUT:

SNO	SNAME	SPHNO
103	ROJA	228603
104	TEJA	228604

3. Display student names who has secured marks greater than that of 'CN'

Select sno, sname from student where sno in (select sno from stumarks where stumarks> (select smarks from stumarks where sno in (select sno from stucourse where scourse='CN')))

OUTPUT:

SNO	SNAME	SPHNO
102	PUJA	228602

4.Display student who has not enrolled dbms

Select \* from student where sno not in(select sno from stucourse where scourse='DBMS')

OUTPUT:

SNO	SNAME	SPHNO
103	SUSHMA	228603

## • Complex Queries

Create the following tables

Account (acno,branchname,balance) acno-primary key

Customer(cname,custstreet,custcity) cname-primary key

Loan(lno,branchname,amount) lno-primary key

Depositor(cname,acno) acno-foreign key Account(acno)

Borrower(cname,lno) lno-foreign key Loan(lno)

Here relationship between the two tables in the given below

Depositor&Account,Borrower&Loan,Customer&Depositor

1. Find customer names whose city is Harrison.

A: select cust\_name from customer where cust\_city='harrison'



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)



Estd : 2008

OUTPUT:

Accreated by NAAC with A+ and NBA  
Affiliated to Osmania University & Approved by AICTE

Cname  
Hayes  
Jones





2. Find names of all bank customers who have either an account or a loan or both.

A: select cust\_name from borrower union select cust\_name from depositor.

OUTPUT:

```
Cname
Adams
Curry
Hayes
Jackson
Johnson
Jones
```

To view even duplicate values, use:

select cust\_name from borrower union all select cust\_name from depositor

3. Find all customers who have both a loan and an account.

A: select cust\_name from borrower intersect select cust\_name from depositor.

OUTPUT:

```
Cname
Hayes
Jones
smith
```

4. Find the names of all customers who have a loan at Perryridge branch.

A: Using Nested queries:

select cust\_name from borrower where lno in (select lno from loan where branchname='perryridge')

Using joins:

select cust\_name from borrower b, loan l where b.lno=l.lno and l.branchname='perryridge'.

OUTPUT:

```
Cname
Adams
Hayes
```

5. Find all customers who have an account but no loan at the bank.

A: select cust\_name from depositor minus select cust\_name from borrower.

OUTPUT:

```
Cname
Johnson
Lindsay
Tuner
```

6. Find the no of depositors for each branch

A: select branch\_name,count(distinct customer\_name) from depositor,account where depositor.acno=account.acno group by branch\_name

7. Find the average balance for each customer who lives in Harrison and has atleast 3 accounts

A: select depositor.customer\_name,avg(balance) from depositor,account,customer Where depositor.acno=account.acno and depositor.customer\_name=customer.customer\_name and customer\_city='Harrison' group by depositor.customer\_name having count(distinct depositor.acno)>=3.



Estd : 2008

**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

**Accredited by NAAC with A+ and NBA**

**Affiliated to Osmania University & Approved by AICTE**



### 3.AIM: CREATING TABLE USING COMBINATION OF CONSTRAINTS

#### **MYSQL CONSTRAINTS:**

Constraints are used to limit the type of data that can go into a table.

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

We will focus on the following constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

The next chapters will describe each constraint in detail.

#### **a) MYSQL NOT NULL Constraint**

The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

##### **Example:**

```
CREATE TABLE Persons  
( P_Id int NOT NULL)
```

#### **b) MYSQL UNIQUE Constraint**

The UNIQUE constraint uniquely identifies each record in a database table.

The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column

or set of columns. A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

##### **Example:**

```
CREATE TABLE Persons  
(  
P_Id int NOT NULL UNIQUE  
)
```

#### **c)MYSQL PRIMARY KEY Constraint**

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain unique values. A primary key column cannot contain NULL values.

Each table should have a primary key, and each table can have only ONE primary key.

##### **Example:**

```
CREATE TABLE Persons  
(  
P_Id int NOT NULL,  
Name varchar2(10), Primary key(P_Id)  
)
```



Estd : 2008

**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

**Accredited by NAAC with A+ and NBA**

**Affiliated to Osmania University & Approved by AICTE**



**d) MYSQL FOREIGN KEY Constraint**

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

**Example:**

```
CREATE TABLE Orders
(
  O_Id int NOT NULL,
  OrderNo int NOT NULL,
  P_Id int,
  PRIMARY KEY (O_Id),
  FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

**e) MYSQL CHECK Constraint**

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table, it can limit the values in certain columns based on values

in other columns in the row.

**Example:**

```
CREATE TABLE Persons
(
  P_Id int NOT NULL,
  CHECK (P_Id>0)
)
```

**f)MYSQL DEFAULT Constraint**

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

**Example:**

```
CREATE TABLE Persons
(
  P_Id int NOT NULL,
  P_NAME VARCHAR2(10)
  DEFAULT 'SWAPNA')
```

**4.AIM :EXERCISING TYPES OF JOINS****MYSQL JOIN**

The JOIN keyword is used in an MYSQL statement to query data from two or more tables, based on a relationship between certain columns in these tables. Tables in a database are often related to each other with keys. A primary key is a column (or a combination of columns) with a unique value for each row. Each primary key value must be unique within the table. The purpose is to bind data together, across tables, without repeating all of the data in every table.

Look at the "Persons" table:

<u>P_Id</u>	<u>LastName</u>	<u>FirstName</u>	<u>Address</u>	<u>City</u>
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger



Note that the "P\_Id" column is the primary key in the "Persons" table. This means that **no** two rows can have the same P\_Id. The P\_Id distinguishes two persons even if they have the same name.

Next, we have the "Orders" table:

<u>O_Id</u>	<u>OrderNo</u>	<u>P_Id</u>
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Note that the "O\_Id" column is the primary key in the "Orders" table and that the "P\_Id" column refers to the persons in the "Persons" table without using their names. Notice that the relationship between the two tables above is the "P\_Id" column.

### MYSQL JOINS:

- **JOIN** : Return rows when there is at least one match in both tables.
- **LEFT JOIN** : Return all rows from the left table, even if there are no matches in the right table.
- **RIGHT JOIN** : Return all rows from the right table, even if there are no matches in the left table.
- **FULL JOIN** : Return rows when there is a match in one of the tables.

### MYSQL INNER JOIN Keyword

The INNER JOIN keyword return rows when there is at least one match in both tables.

### MYSQL INNER JOIN

#### Syntax

SELECT column\_name(s) FROM table\_name1 INNER JOIN table\_name2 ON  
table\_name1.column\_name=table\_name2.column\_name

**PS:** INNER JOIN is the same as JOIN.

### MYSQL INNER JOIN Example

The "Persons" table:

<u>P_Id</u>	<u>LastName</u>	<u>FirstName</u>	<u>Address</u>	<u>City</u>
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

<u>O_Id</u>	<u>OrderNo</u>	<u>P_Id</u>
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15



Now we want to list all the persons with any orders.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons
INNER JOIN Orders ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName
```

The result-set will look like this:

<u>LastName</u>	<u>FirstName</u>	<u>OrderNo</u>
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

The INNER JOIN keyword return rows when there is at least one match in both tables. If there are rows in "Persons" that do not have matches in "Orders", those rows will NOT be listed.

### MYSQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table\_name1), even if there are no matches in the right table (table\_name2).

### MYSQL LEFT JOIN

#### Syntax:

```
SELECT column_name(s) FROM table_name1 LEFT JOIN table_name2 ON
table_name1.column_name=table_name2.column_name
```

**PS:** In some databases LEFT JOIN is called LEFT OUTER JOIN.

### MYSQL LEFT JOIN Example

The "Persons" table:

<u>P_Id</u>	<u>LastName</u>	<u>FirstName</u>	<u>Address</u>	<u>City</u>
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

<u>O_Id</u>	<u>OrderNo</u>	<u>P_Id</u>
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons and their orders - if any, from the tables above.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons
LEFT JOIN Orders ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName
```

The result-set will look like this:

<u>LastName</u>	<u>FirstName</u>	<u>OrderNo</u>
Hansen	Ola	22456



Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	34764

The LEFT JOIN keyword returns all the rows from the left table (Persons), even if there are no matches in the right table (Orders).

### MYSQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all the rows from the right table (table\_name2), even if there are no matches in the left table (table\_name1).

### MYSQL RIGHT JOIN Syntax

SELECT column\_name(s) FROM table\_name1 RIGHT JOIN table\_name2 ON  
table\_name1.column\_name=table\_name2.column\_name

**PS:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

### MYSQL RIGHT JOIN Example

The "Persons" table:

<u>P_Id</u>	<u>LastName</u>	<u>FirstName</u>	<u>Address</u>	<u>City</u>
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

<u>O_Id</u>	<u>OrderNo</u>	<u>P_Id</u>
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the orders with containing persons - if any, from the tables above.

We use the following SELECT statement:

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons RIGHT JOIN  
Orders ON Persons.P\_Id=Orders.P\_Id ORDER BY Persons.LastName

The result-set will look like this:

<u>LastName</u>	<u>FirstName</u>	<u>OrderNo</u>
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

The RIGHT JOIN keyword returns all the rows from the right table (Orders), even if there are no matches in the left table (Persons).

### MYSQL FULL JOIN Keyword

The FULL JOIN keyword return rows when there is a match in one of the tables.





## MYSQL FULL JOIN

### Syntax

```
SELECT column_name(s) FROM table_name1 FULL JOIN table_name2 ON
table_name1.column_name=table_name2.column_name
```

### MYSQL FULL JOIN Example

The "Persons" table:

<u>P_Id</u>	<u>LastName</u>	<u>FirstName</u>	<u>Address</u>	<u>City</u>
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

<u>O_Id</u>	<u>OrderNo</u>	<u>P_Id</u>
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Now we want to list all the persons and their orders, and all the orders with their persons.

We use the following SELECT statement:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo FROM Persons
FULL JOIN Orders ON Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName
```

The result-set will look like this:

<u>LastName</u>	<u>FirstName</u>	<u>OrderNo</u>
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	34764

The FULL JOIN keyword returns all the rows from the left table (Persons), and all the rows from the right table (Orders). If there are rows in "Persons" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Persons", those rows will be listed as well.



## EXPERIMENT

### 8.AIM: USAGE OF TRIGGERS & STORED PROCEDRES

A trigger is a set of actions that are run automatically when a specified change operation is performed on a specified table. The change operation can be an MYSQL INSERT, UPDATE, or DELETE statement, or an insert, update, or delete high level language statement in an application program. Triggers are useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail.

Triggers can be defined in two different ways:

- MYSQL triggers
- External triggers

For an **external trigger**, the CRTPFTRG CL command is used. The program containing the set of trigger actions can be defined in any supported high level language. External triggers can be insert, update, delete, or read triggers.

For an **MYSQL trigger**, the CREATE TRIGGER statement is used. The trigger program is defined entirely using MYSQL. MYSQL triggers can be insert, update, or delete triggers.

Once a trigger is associated with a table, the trigger support calls the trigger program whenever a change operation is initiated against the table, or any logical file or view created over the table. MYSQL triggers and external triggers can be defined for the same table. Up to 200 triggers can be defined for a single table.

Each change operation can call a trigger before or after the change operation occurs. Additionally, you can add a read trigger that is called every time the table is accessed. Thus, a table can be associated with many types of triggers.

- Before delete trigger
- Before insert trigger
- Before update trigger
- After delete trigger
- After insert trigger
- After update trigger
- Read-only trigger (external trigger only)

#### Examples:

##### 1. Create trigger for before insertion

```
MYSQL> create table orders (order_id number (10), quantitynumber (10),cost_per_item  
numeric(6,2),total_cost numeric(8,2),create_datedate,created_by varchar2(20));
```

Table created.

```
create or replace trigger o_before_insert
```

```
before insert
```

```
on orders
```

```
for each row
```

```
declare
```

```
v_username varchar2(20);
```

```
begin
```

```
select user into v_username from dual;
```

```
:new.create_date:=sysdate;
```

```
:new.created_by:=v_username;
```

```
* end;
```

```
MYSQL> /
```

Trigger created.



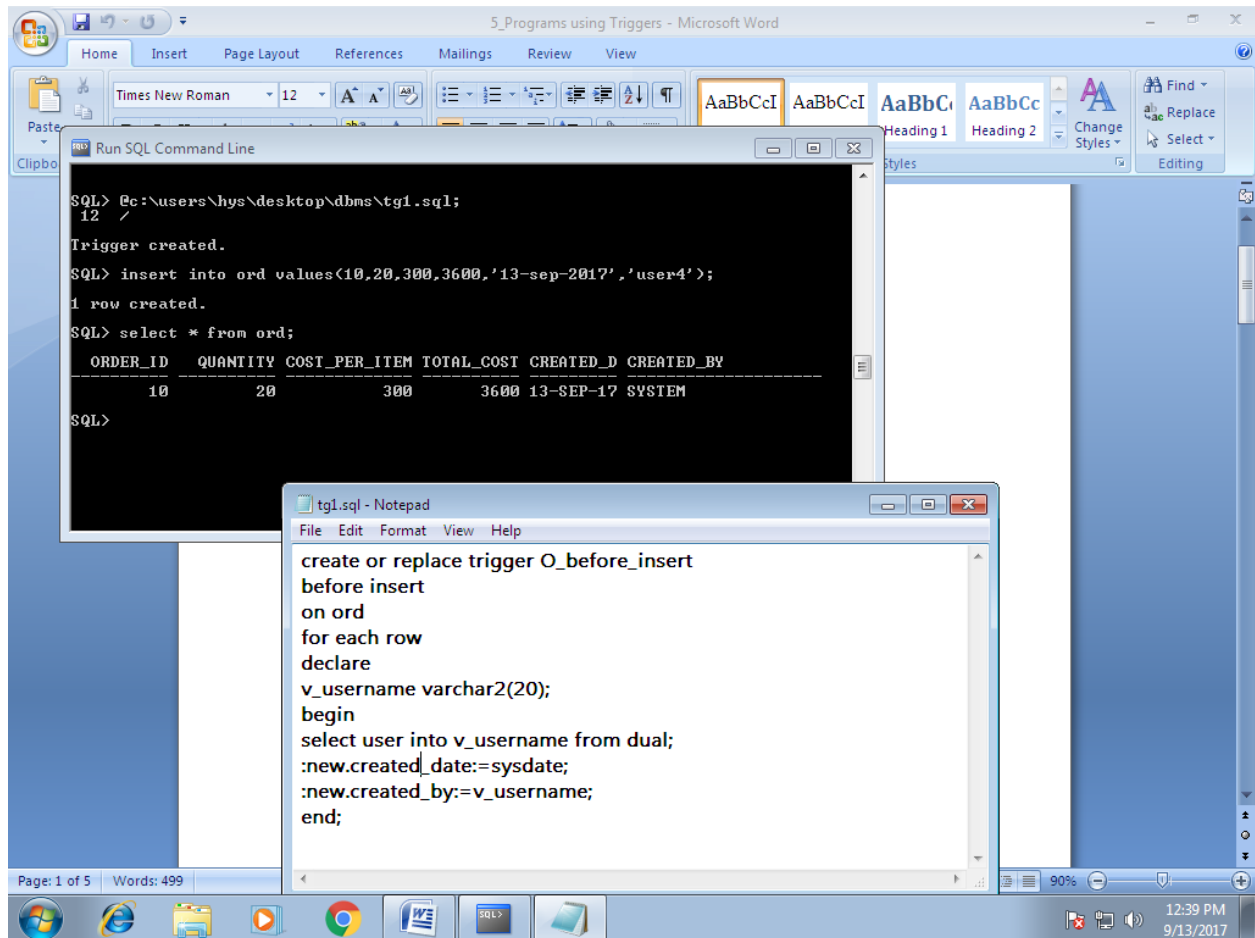
To Check the created trigger:

MYSQL> insert into orders values(10,20,30,40,'23-jun-2006','user4');

1 row created.

MYSQL> select \* from orders;

ORDER_ID	QUANTITY	COST_PER_ITEM	TOTAL_COST	CREATE_DA	CREATED_BY
10	20	30	40	21-AUG-07	UNIV04



## 2. Create Trigger for after insertion:

MYSQL> create table order4(order\_id number(10),quantity number(10),cost\_per\_item numeric(6,2),total\_cost numeric(8,2));

Table created.

MYSQL> create table orders\_audit(order\_id number(10),quantity number(10),cost\_per\_item numeric(6,2),total\_cost numeric(8,2),username varchar2(20));

Table created.

create or replace trigger orders\_after\_insert

after insert

on order4

for each row

declare

v\_username varchar2(20);

begin

select user into v\_username from dual;

insert into orders\_audit(order\_id,quantity,cost\_per\_item,total\_cost,username)

values(:new.order\_id,:new.quantity,:new.cost\_per\_item,:new.total\_cost,v\_username);

10 end;



MYSQL> /

Trigger created.

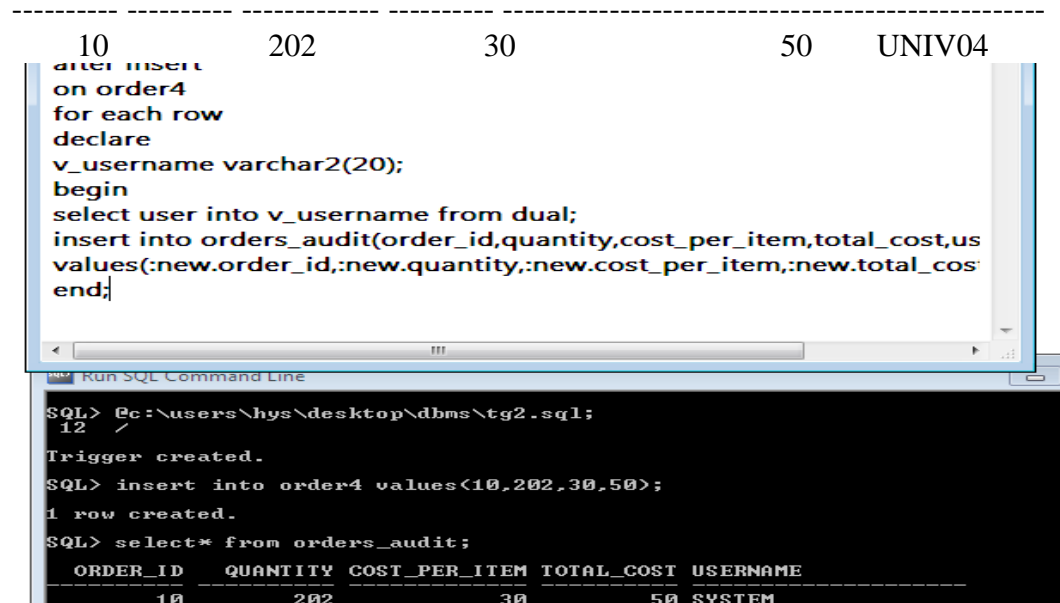
To Check the created Trigger:

MYSQL> insert into order4 values(10,202,30,50);

1 row created.

MYSQL> select \* from orders\_audit;

ORDER_ID	QUANTITY	COST_PER_ITEM	TOTAL_COST	USERNAME
10	202	30	50	UNIV04



### 3. Create a trigger for before update:

MYSQL> create table order1(order\_id number(10),quantity number(10),cost\_per\_item numeric(6,2),total\_cost numeric(8,2),updated\_date date,updated\_by varchar2(20));

Table created.

create or replace trigger orders\_before\_update

before update

on order1

for each row

declare

v\_username varchar2(20);

begin

select user into v\_username from dual;

:new.updated\_date:=sysdate;

:new.updated\_by:=v\_username;

\* end;

MYSQL> /

Trigger created.

To Check the created Trigger:

MYSQL> insert into order1 values(10,202,30,50,'23-jul-2006','user4');

1 row created.

MYSQL> update order1 set total\_cost=1000 where order\_id=10;

1 row updated.

MYSQL> select \* from order1;

ORDER_ID	QUANTITY	COST_PER_ITEM	TOTAL_COST	UPDATED_D	UPDATED_BY
10	202	30	1000	21-AUG-07	UNIV04





**4. Create a Trigger for after update:**

```
MYSQL> create table order2(order_id number(10),quantity number(10),cost_per_item
numeric(6,2),total_cost numeric(8,2));
```

Table created.

```
MYSQL> create table orders_audit1(order_id number(10),quantity_before
number(10),quantity_after number(10),username varchar2(20));
```

Table created.

```
create or replace trigger orders_after_update
after update
on order2
for each row
declare
v_username varchar2(20);
begin
select user into v_username from dual;
insert into orders_audit1(order_id,quantity_before,quantity_after,username)
values(:new.order_id,:old.quantity,:new.quantity,v_username);
* end;
```

```
MYSQL> /
```

Trigger created.

To Check the created Trigger:

```
MYSQL> insert into order2 values(10,202,30,50);
```

1 row created.

```
MYSQL> update order2 set quantity=100 where order_id=10;
```

1 row updated.

```
MYSQL> select * from orders_audit1;
```

```
ORDER_ID QUANTITY_BEFORE QUANTITY_AFTER USERNAME
```

```
-----
-----
10          202          100          UNIV04
```

**5. Create a Trigger for after delete:**

```
MYSQL> create table order3(order_id number(10),quantity number(10),cost_per_item
numeric(6,2),total_cost numeric(8,2));
```

Table created.

```
MYSQL> create table orders_audit2(order_id number(10),quantity number(10),cost_per_item
number(10),total_cost number(10),delete_date date,deleted_by varchar2(20));
```

Table created.

```
create or replace trigger orders_after_delete
after delete
on order3
for each row
declare
v_username varchar2(20);
begin
select user into v_username from dual;
insert into orders_audit2(order_id,quantity,cost_per_item,total_cost,delete_date,deleted_by)
values(:old.order_id,:old.quantity,:old.cost_per_item,
:old.total_cost,sysdate,v_username);
*end;
```

```
MYSQL> / Trigger created.
```



To Check the output:

MYSQL> insert into order3 values(10,2,90,50);

1 row created.

MYSQL> insert into order3 values(10,20,300,450);

1 row created.

MYSQL> delete order3 where order\_id=10;

2 rows deleted.

MYSQL> select \* from orders\_audit2;

<i>ORDER</i>	<i>ID</i>	<i>QUANTITY</i>	<i>COST PER ITEM</i>	<i>TOTAL COST</i>	<i>DELETE DATE</i>	<i>DELETED BY</i>
10	2	90	50	21-AUG-07	UNIV04	
10	20	300	450	21-AUG-07	UNIV04	

## 7.AIM: WRITING PL/SQL BLOCKS

### Introduction

- PL/SQL is Oracle's **procedural** language extension to MYSQL, the non-procedural relational database language.
- With PL/SQL, you can use MYSQL statements to manipulate ORACLE data and the flow of control statements to process the data. Moreover, you can declare constants and variables, define subprograms (procedures and functions), and trap runtime errors. Thus, PL/SQL combines the data manipulating power of MYSQL with the data processing power of procedural languages. PL/SQL blocks contain three sections
- Declare section
- Executable section and
- Exception-handling section.

The executable section is the only mandatory section of the block.

Both the declaration and exception-handling sections are optional.

PL/SQL block has the following structure:

```

DECLARE
    Declaration statements
BEGIN
    Executable statements
EXCETION
    Exception-handling statements
END;
```

1) **Aim: Write a PL/SQL block to Perform Arithmetic Operations on the given Two numbers**

**Input:**

declare

a number(5);

b number(5);

c number(5);

begin

a:=a;

b:=b;

c:=a+b;

dbms\_output.put\_line('Sum of Two numbers is:'|| c );

c:=a-b;

dbms\_output.put\_line('Difference of Two Numbers is:'|| c);



```
c:=a*b;  
dbms_output.put_line('Product of Two numbers is:'|| c);  
c:=a/b;  
dbms_output.put_line('Division of Two numbers is:'|| c);  
end;
```

**Output:**

A=62

B=45

Sum of two numbers is: 107

Difference of two numbers is: 17

Product of two numbers is :2790

Division of two numbers is :1.5

**2) Aim: Write a PL/SQL Program to Find Given Number is Even Or Not**

**Input:**

```
DECLARE
```

```
Num number(8);
```

```
BEGIN
```

```
Num:=: Num;
```

```
--if the number is divisible by 2 then it is a even number.
```

```
IF MOD (num, 2) =0 THEN
```

```
--display the number.
```

```
dbms_output.put_line(num||'is Even');
```

```
ELSE
```

```
dbms_output.put_line(num||'is odd');
```

```
END IF;
```

```
END;
```

**Output:**

Num 25 is odd

**3)Aim: Write a PL/SQL block to find the Factorial of a Given Number**

**Input:**

```
DECLARE
```

```
num number(5);
```

```
fact number(5):=1;
```

```
k number(5);
```

```
BEGIN
```

```
num:=:num;
```

```
k:=num;
```

```
while(num>0)
```

```
loop
```

```
fact:=fact*num;
```

```
num:=num-1;
```

```
end loop;
```

```
dbms_output.put_line('FACTORIAL OF '||k||' IS '||fact);
```

```
end;
```

**OUTPUT:** Factorial of 5 is 120

**4. Write a PL/SQL block to Print the Reverse of a Given Number**



Estd : 2008

Accreated by NAAC with A+ and NBA

Affiliated to Osmania University & Approved by AICTE

**Input:**

```
DECLARE
n number(5) := &n;
rev number(3):=0;
s number(5);
BEGIN
  while(n>0)
  loop
s:=mod(n,10);
rev:=(rev*10)+s;
n:=trunc(n/10);
end loop;
dbms_output.put_line('REVERSE OF a number is'||rev);
end;
```

**OUTPUT:**

N is 987

Reverse of a number is 789

**5. Using PL/SQL find the PALINDROME OF A NUMBER**

```
DECLARE
N number(3):=n;
N1 number(3):=n;
Rev number(5):=0;
R number(5);
Begin
While(n>0)
Loop
R:=n mod10
Rev:=rev*10+r;
N:=trunk(n/10);
End loop;
If(n1==rev) then
Dbms_output.put_line('given number is palindrome');
Else
Dbms_output.put_line('given number is not a palindrome');
End if;
End;
```

**OUTPUT:**

N=626

Given number is palindrome

**6)Using PL/SQL find the given number multiplication table**

**Input:**

```
DECLARE
A number(3):=:a;
C number(3);
I number(3);
Begin
For i in 1..5
```



```
Loop
C:=a*I;
Dbms_output.put_line(a||'*'||i||'=||c);
End loop;
End
OUTPUT:
A=6
6*1=6
6*2=12
6*3=18
6*4=24
6*5=30
```

## AIM: PLMYSQL USING PROCDEURES

### PROCEDURES

A procedure (often called a stored procedure) is a program that can be called to perform operations that can include both host language statements and MYSQL statements.

#### Defining an MYSQL procedure

The CREATE PROCEDURE statement for MYSQL procedures:

- Names the procedure
- Defines the parameters and their attributes
- Provides other information about the procedure which will be used when the procedure is called
- Defines the procedure body. The procedure body is the executable part of the procedure and is a single MYSQL statement.

#### Example:

##### Insert Procedure:

```
create or replace procedure addstudent(no student.%type,name student.sname%type,phno
student.sphno%type) as
begin
insert into student values(5,'ragmani',995684231);
end
```

##### OUTPUT:

Procedure created.

#### TO run the Procedure

```
begin
addstudent(:no,:name,:phno);
end;
```

##### Update Procedure:

```
create or replace procedure updatestudent(no student.%type,name student.sname%type,phno
student.sphno%type) as
begin
update student set sname=name where sno=no;
end;
```

##### OUTPUT:

Procedure created.



### TO run the Procedure

```
begin
update student(:no,:name,:phno);
end;
```

### Delete Procedure:

```
create or replace procedure updatestudent(no student.%type,name student.sname%type,phno
student.sphno%type) as
begin
delete from student where no=sno;
end
```

**OUTPUT:** Procedure created.

### TO run the Procedure

```
begin
delstudent(:no);
end;
```

### Insert procedure :

```
>Create table dee(regno number(10),name varchar2(15),mark1 number(3), mark2 number(3),
mark3 number(3));
Table created.
```

### ins.MYSQL

```
create or replace procedure ins(regno number,name varchar2,mark1 number,mark2 number,mark3
number) is
begin
insert into dee values(regno,name,mark1,mark2,mark3);
end;
>@c:\users\hys\desktop\vai\ins.MYSQL;
/Procedure created.
>exec ins(1001,'vaidehi',90,80,79);
PL/SQL procedure successfully completed.
```

```
>Select * from dee
```

Regno	Name	Mark1	Mark2	Mark3
1001	vaidehi	90	80	79

```
>
```

### First Create a table with eno and name attribute with table name pay.

```
create or replace procedure disp(en in number,c out integer)
as
enum number;
begin
select eno into enum from pay where eno=en;
c:=1;
exception
when no_data_found then
c:=0;
end;
>@C:\user\hys\desktop\vai\disp.MYSQL;
```



>/

PL/SQL Procedure successfully completed.

**The program given below is for calling a procedure (disp)**

declare

p pay%rowtype;

c integer;

begin

p.eno:=&pno;

disp(p.eno,c);

if c=1 then

select eno,name into p.eno,p.name from pay

where eno=p.eno;

dbms\_output.put\_line('eno ||name ');

dbms\_output.put\_line(p.eno ||p.name);

else

dbms\_output.put\_line('error enter the correct emp name');

end if;

end;

>@c:\users\hys1\desktop\vai\caldisp.MYSQL;

>/

Enter value for pno:101

Old 5: p.eno:=&pno;

New 5: p.eno:=103;

Error enter the correct emp name

PL/SQL procedure successfully completed.

>@c:\users\hys1\desktop\vai\caldisp.MYSQL;

>/

Enter value for pno:101

Old 5: p.eno:=&pno;

New 5: p.eno:=101;

eno name

101 vaidehi

PL/SQL procedure successfully completed.

**If the P.eno is not present in the table then it will tell error message or other show the entire tuple of particular eno.**





```

SQL> set serveroutput on;
SQL> @c:\users\hys1\desktop\wai\caldisp.sql;
/
Enter value for pno: 101
old 5: p.eno:=&pno;
new 5: p.eno:=101;
eno name
101vai
PL/SQL procedure successfully completed.

SQL> @c:\users\hys1\desktop\wai\caldisp.sql;
/
Enter value for pno: 103
old 5: p.eno:=&pno;
new 5: p.eno:=103;
error enter the correct emp name
PL/SQL procedure successfully completed.

SQL>
caldisp.sql - Notepad
File Edit Format View Help
create or replace procedure disp(en in number,c out integer)
as
enum number;
begin
select eno into enum from pay where eno=en;
c:=1;
exception
when no_data_found then
c:=0;
end;

caldisp.sql - Notepad
File Edit Format View Help
declare
p pay%rowtype;
c integer;
begin
p.eno:=&pno;
disp(p.eno,c);
if c=1 then
select eno,name into p.eno,p.name from pay
where eno=p.eno;
dbms_output.put_line('eno '||p.name);
dbms_output.put_line(p.eno ||p.name);
else
dbms_output.put_line('error enter the correct emp name');
end if;
end;
  
```

**MYSQL> create or replace procedure pali(str in varchar,rev out varchar) is**

```

c number;
begin
c:=length(str);
while c>0 loop
rev:=rev||substr(str,c,1);
c:=c-1;
end loop;
end;
/
  
```

Procedure created.

```

MYSQL> declare
str varchar2(20):='&str';
rev varchar2(20);
begin
pali(str,rev);
dbms_output.put_line('given string is '||str);
dbms_output.put_line('reversed string is '||rev);
if str=rev then
dbms_output.put_line('palindrome');
else
dbms_output.put_line('not a palindrome');
end if;
end;
/
  
```

Enter value for str: deepa

old 2: str varchar2(20):='&str';

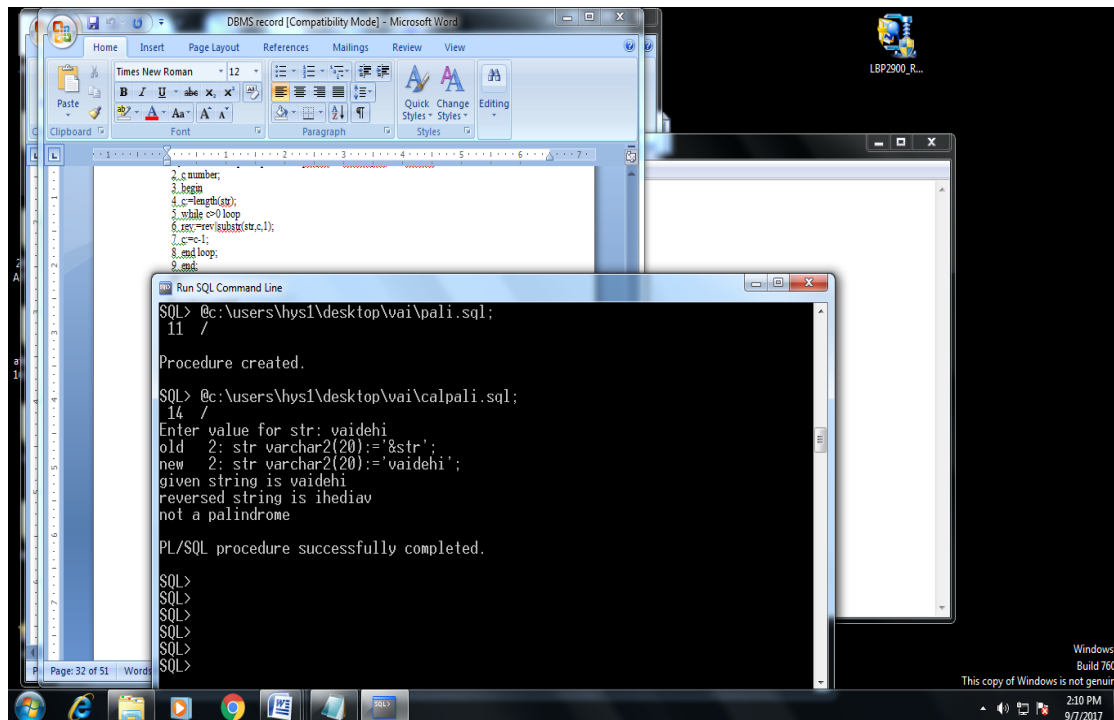
new 2: str varchar2(20):='deepa';

given string is deepa

reversed string is apeed

not a palindrome

PL/SQL procedure successfully completed.



## EXPERIMENT

### 9.AIM :CREATION OF FORMS

#### Introduction

Use Form Builder to simplify for the creation of data-entry screens, also known as Forms. Forms are the applications that connect to a database, retrieve information requested by the user, present it in a layout specified by Form designer, and allow the user to modify or add information. Form Builder allows you to build forms quickly and easily

```
import java.awt.*;  
import java.applet.*;  
import java.awt.event.*;  
public class student extends Frame implements ActionListener  
{String msg;  
    Button b1=new Button("save");  
    Label l11=new Label("Student details",Label.CENTER);
```



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

**Accreated by NAAC with A+ and NBA**

Estd : 2008    Affiliated to Osmania University & Approved by AICTE



```
Label l1=new Label("Name:",Label.LEFT);
Label l2=new Label("year:",Label.LEFT);
Label l3=new Label("Sex(M/F):",Label.LEFT);
Label l4=new Label("Address:",Label.LEFT);
Label l5=new Label("Course:",Label.LEFT);
Label l6=new Label("Semester:",Label.LEFT);
Label l7=new Label("",Label.RIGHT);
TextField t1=new TextField();
Choice c1=new Choice();
CheckboxGroup cbg=new CheckboxGroup();
Checkbox ck1=new Checkbox("Male",false,cbg);
Checkbox ck2=new Checkbox("Female",false,cbg);
TextArea t2=new TextArea("",180,90,TextArea.SCROLLBARS_VERTICAL_ONLY);
Choice course=new Choice();
Choice sem=new Choice();
Choice year=new Choice();
public student()
{addWindowListener(new myWindowAdapter());
setBackground(Color.cyan);
setForeground(Color.black);
setLayout(null);
add(l1);
add(l2);
add(l3);
add(l4);
add(l5);
add(l6);
add(l7);
add(t1);
add(t2);
add(ck1);
add(ck2);
add(course);
add(sem);
```



```
add(year);
add(b1);
b1.addActionListener(this);
add(b1);
course.add("BE CSE");
course.add("BE IT");
course.add("BE ECE");
course.add("BE EEE");
course.add("MBA");
sem.add("1");
sem.add("2");
sem.add("3");
sem.add("4");
sem.add("5");
sem.add("6");
year.add("I");
year.add("II");
year.add("III");
year.add("IV");
// age.add("21");
l1.setBounds(25,65,90,20);
l2.setBounds(25,90,90,20);
l3.setBounds(25,120,90,20);
l4.setBounds(25,185,90,20);
l5.setBounds(25,260,90,20);
l6.setBounds(25,290,90,20);
l7.setBounds(25,260,90,20);
l11.setBounds(10,40,280,20);
t1.setBounds(120,65,170,20);
t2.setBounds(120,185,170,60);
ck1.setBounds(120,120,50,20);
ck2.setBounds(170,120,60,20);
course.setBounds(120,260,100,20);
sem.setBounds(120,290,50,20);
year.setBounds(120,90,50,20);
b1.setBounds(120,350,50,30);
}
public void paint(Graphics g)
{g.drawString(msg,200,450);}
public void actionPerformed(ActionEvent ae)
{if(ae.getActionCommand().equals("save"))
 {msg="Student details saved!";
  setForeground(Color.red); }
}
public static void main(String g[])
{student stu=new student();
 stu.setSize(new Dimension(500,500));
 stu.setTitle("student registration");
 stu.setVisible(true);
}
}
```

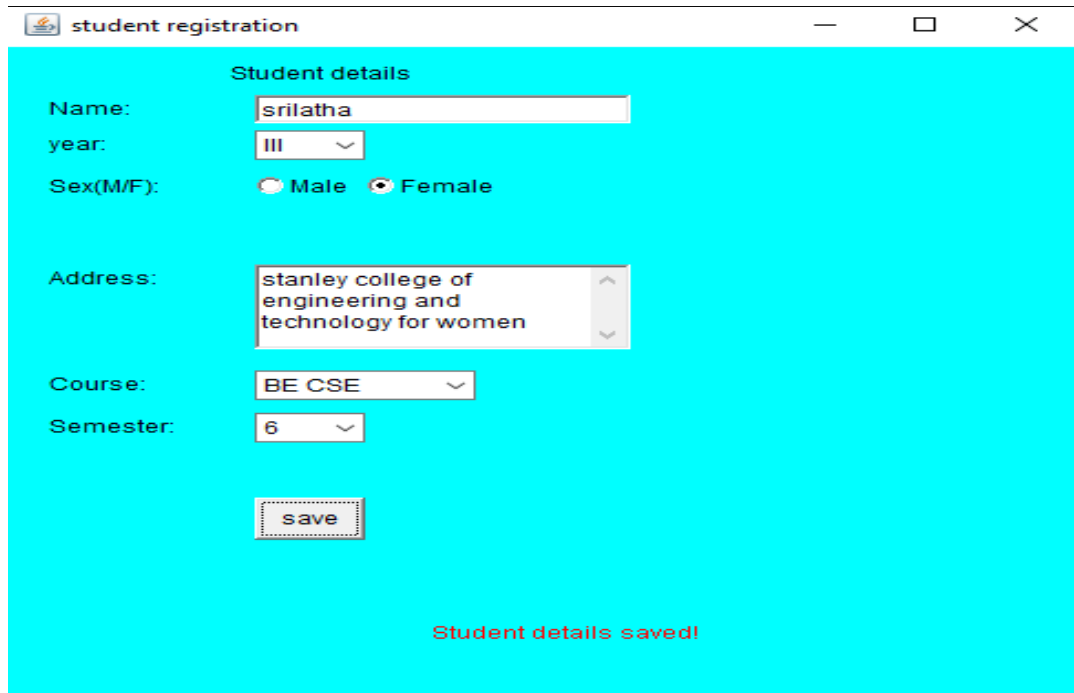


```
class myWindowAdapter extends WindowAdapter
{public void windowClosing(WindowEvent we) {
    System.exit(0);
}
}
```

output:

Javac student.java

java student



The screenshot shows a Java Swing window titled "student registration". The window has a cyan background and contains the following elements:

- Student details** (Section Header)
- Name:** A text field containing "srilatha".
- year:** A dropdown menu showing "III".
- Sex(M/F):** Two radio buttons, "Male" (unselected) and "Female" (selected).
- Address:** A text area containing "stanley college of engineering and technology for women".
- Course:** A dropdown menu showing "BE CSE".
- Semester:** A dropdown menu showing "6".
- save** (Button)
- Student details saved!** (Confirmation message)





## EXPERIMENT

### 8.AIM: PL/SQL USING PROCEDURE FOR DATA VALIDATION

#### PROCEDURES

A procedure (often called a stored procedure) is a program that can be called to perform operations that can include both host language statements and MYSQL statements.

#### Defining an MYSQL procedure

The CREATE PROCEDURE statement for MYSQL procedures:

- Names the procedure
- Defines the parameters and their attributes
- Provides other information about the procedure which will be used when the procedure is called
- Defines the procedure body. The procedure body is the executable part of the procedure and is a single MYSQL statement.

#### Example:

##### Insert Procedure:

```
create or replace procedure addstudent(no student.%type,name student.sname%type,phno
student.sphno%type) as
begin
insert into student values(5,'ragmani',995684231);
end
```

##### OUTPUT:

Procedure created.

##### TO run the Procedure

```
begin
addstudent(:no,:name,:phno);
end;
```

##### Update Procedure:

```
create or replace procedure updatestudent(no student.%type,name student.sname%type,phno
student.sphno%type) as
begin
update student set sname=name where sno=no;
end;
```

##### OUTPUT:

Procedure created.

##### TO run the Procedure

```
begin
update student(:no,:name,:phno);
end;
```

##### Delete Procedure:

```
create or replace procedure updatestudent(no student.%type,name student.sname%type,phno
student.sphno%type) as
begin
delete from student where no=sno;
end
```

**OUTPUT:** Procedure created.



### TO run the Procedure

```
begin
delstudent(:no);
end;
```

#### Insert procedure :

```
>Create table dee(regno number(10),name varchar2(15),mark1 number(3), mark2 number(3), mark3
number(3));
Table created.
```

create or replace procedure ins(regno number,name varchar2,mark1 number,mark2 number,mark3 number) is

```
begin
insert into dee values(regno,name,mark1,mark2,mark3);
end;
```

```
>@c:\users\hys\desktop\vai\ins.SQL;
```

/Procedure created.

```
>exec ins(1001,'vaidehi',90,80,79);
```

PL/SQL procedure successfully completed.

```
>Select * from dee
```

Regno	Name	Mark1	Mark2	Mark3
1001	vaidehi	90	80	79

```
>
```

#### 2) First Create a table with eno and name attribute with table name pay.

create or replace procedure disp(en in number,c out integer)

as

```
enum number;
```

```
begin
```

```
select eno into enum from pay where eno=en;
```

```
c:=1;
```

```
exception
```

```
when no_data_found then
```

```
c:=0;
```

```
end;
```

```
>@C:\user\hys\desktop\vai\disp.MYSQL;
```

```
>/
```

PL/SQL Procedure successfully completed.

#### The program given below is for calling a procedure (disp)

```
declare
```

```
p pay%rowtype;
```

```
c integer;
```

```
begin
```

```
p.eno:=&pno;
```

```
disp(p.eno,c);
```

```
if c=1 then
```

```
select eno,name into p.eno,p.name from pay
```

```
where eno=p.eno;
```

```
dbms_output.put_line('eno '||name ');
```

```
dbms_output.put_line(p.eno ||p.name);
```

```
else
```

```
dbms_output.put_line('error enter the correct emp name');
```



Estd : 2008

end if;

end;

>@c:\users\hys1\desktop\vai\caldisp.MYSQL;

>/

Enter value for pno:101

Old 5: p.eno:=&pno;

New 5: p.eno:=103;

Error enter the correct emp name

PL/SQL procedure successfully completed.

>@c:\users\hys1\desktop\vai\caldisp.MYSQL;

>/

Enter value for pno:101

Old 5: p.eno:=&pno;

New 5: p.eno:=101;

eno name

101 vaiidehi

PL/SQL procedure successfully completed.

If the P.eno is not present in the table then it will tell error message or other show the entire tuple of particular eno.

```

SQL> set serveroutput on;
SQL> @c:\users\hys1\desktop\vai\caldisp.sql;
15 /
Enter value for pno: 101
old 5: p.eno:=&pno;
new 5: p.eno:=101;
eno name
101vai

PL/SQL procedure successfully completed.

SQL> @c:\users\hys1\desktop\vai\caldisp.sql;
15 /
Enter value for pno: 103
old 5: p.eno:=&pno;
new 5: p.eno:=103;
error enter the correct emp name

PL/SQL procedure successfully completed.

SQL>
  
```

```

create or replace procedure disp(en in number,c out integer)
as
enum number;
begin
select eno into enum from pay where eno=en;
c:=1;
exception
when no_data_found then
c:=0;
end;
  
```

```

declare
p pay%rowtype;
c integer;
begin
p.eno:=&pno;
disp(p.eno,c);
if c=1 then
select eno,name into p.eno,p.name from pay
where eno=p.eno;
dbms_output.put_line('eno '||p.name);
dbms_output.put_line(p.eno ||p.name);
else
dbms_output.put_line('error enter the correct emp name');
end if;
end;
  
```

3)

MYSQL> create or replace procedure pali(str in varchar,rev out varchar) is

2 c number;

3 begin

4 c:=length(str);

5 while c>0 loop

6 rev:=rev||substr(str,c,1);

7 c:=c-1;

8 end loop;

9 end;



Procedure created.

MYSQL> declare

```
2 str varchar2(20):='&str';
3 rev varchar2(20);
4 begin
5 pali(str,rev);
6 dbms_output.put_line('given string is '||str);
7 dbms_output.put_line('reversed string is '||rev);
8 if str=rev then
9 dbms_output.put_line('palindrome');
10 else
11 dbms_output.put_line('not a palindrome');
12 end if;
13 end;
14 /
```

Enter value for str: deepa

old 2: str varchar2(20):='&str';

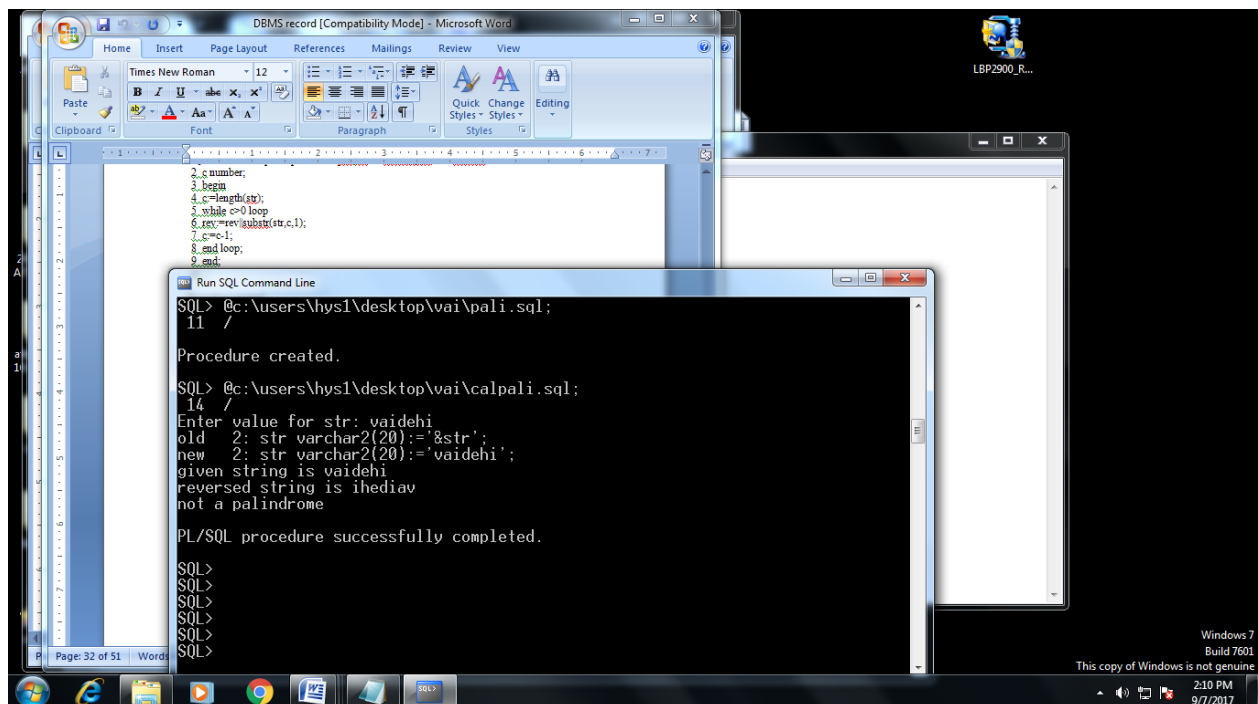
new 2: str varchar2(20):='deepa';

given string is deepa

reversed string is apeed

not a palindrome

PL/SQL procedure successfully completed.



## EXPERIMENT

### 9.AIM : REPORT GENERATION BY USING PLSQL REPORTS

#### Introduction

Tabular report shows data in a table format. It is similar in concept to the idea of an Oracle table. Oracle, by default, returns output from your select statement in tabular format.

#### Steps to Create Report in Oracle Reports Builder 10g

##### Step 1: Invoking Reports Builder and the Report Wizard

Welcome to



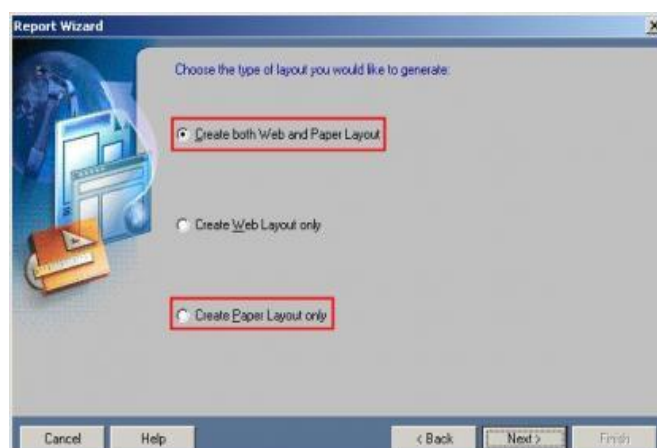
Reports Builder



##### Step 2: Choosing the Layout Type

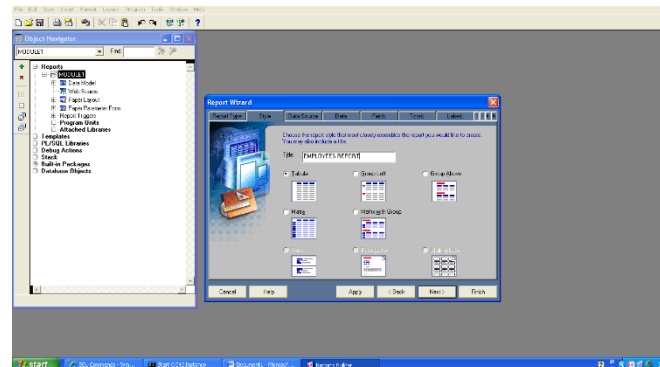
Here you have to specify the type of layout you want the Wizard to generate. The available options are:

- Web and Paper Layout
- Web Layout only
- Paper Layout only

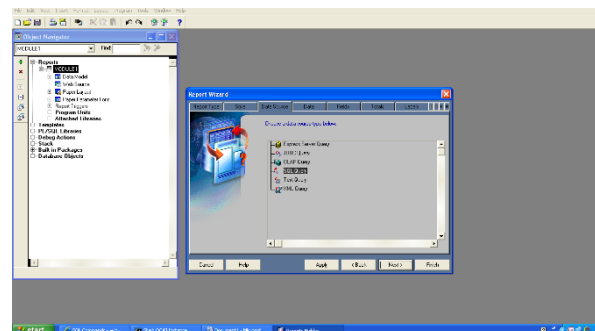


### Step 3: Choosing a Report Style

This page of the Report Wizard shows the various styles of reports. Select Tabular and then click Next.



### Step 4: Selecting the Data Source Type



### Step 5: Building a Query using Query Builder

Building your query with the Query Builder GUI saves you time and increases the ease of use for developers not familiar with building MYSQL statements or with the application tables.

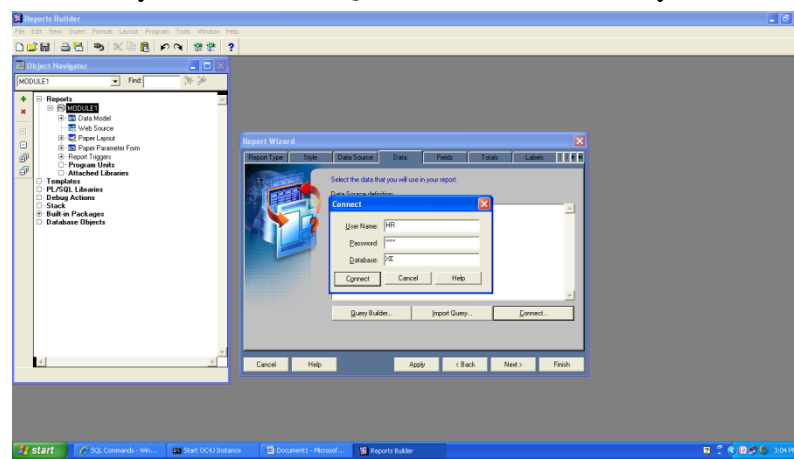
To build a query using Query Builder:

1. Select Query Builder from the Query page in the Report Wizard.
2. Enter your username, password, and alias in the Connect dialog box that appears if you have not already connected to the database.
3. Select the data tables to build the query.
4. Click Include. The tables appear in the selection area.
5. Click Close to close the Select Data Tables window.
6. In each table, double-click the column names that you want in the query, or use the check boxes. To select all columns, double-click the Table title bar.
7. Click OK.

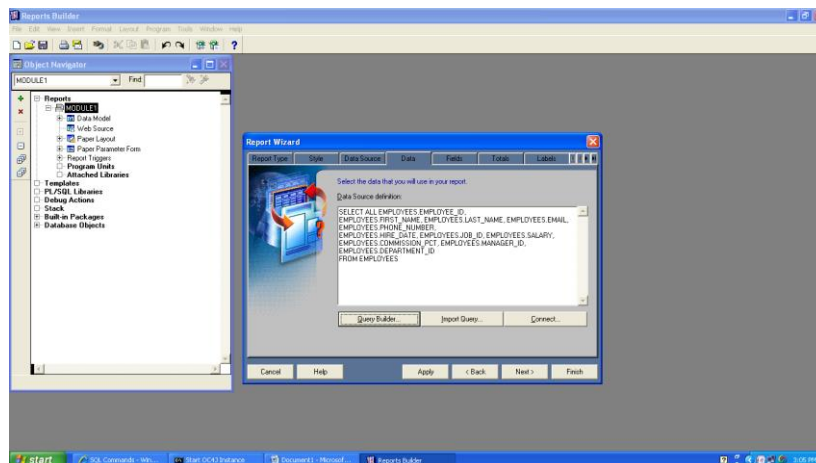
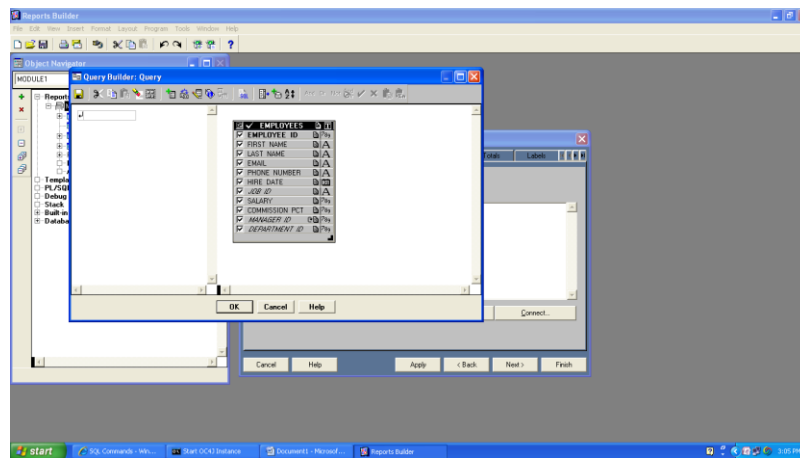
Query Builder copies the query syntax into the Report Wizard. You can modify the query by reentering Query Builder or by modifying the MYSQL query statement text.

**Note:** If you prefer to write your own MYSQL statement, enter the syntax directly in the MYSQL query statement

area of the Query Alternatively, you can import the contents of a file by clicking Import MYSQL



area of the Query Alternatively, you can import the contents of a file by clicking Import MYSQL



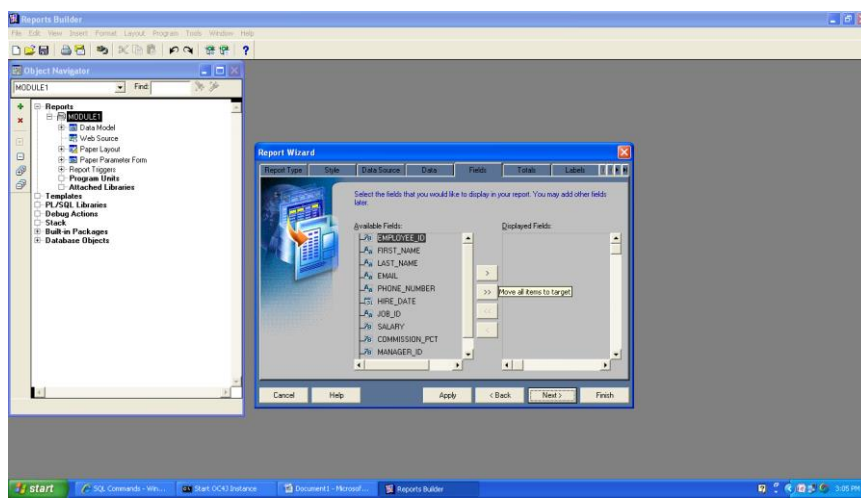
## Step 6: Selecting Displayed Fields

In the Field page, select each field from the Available Fields list and click >. The selected fields move to the Displayed Fields list. To display all fields, click >>.

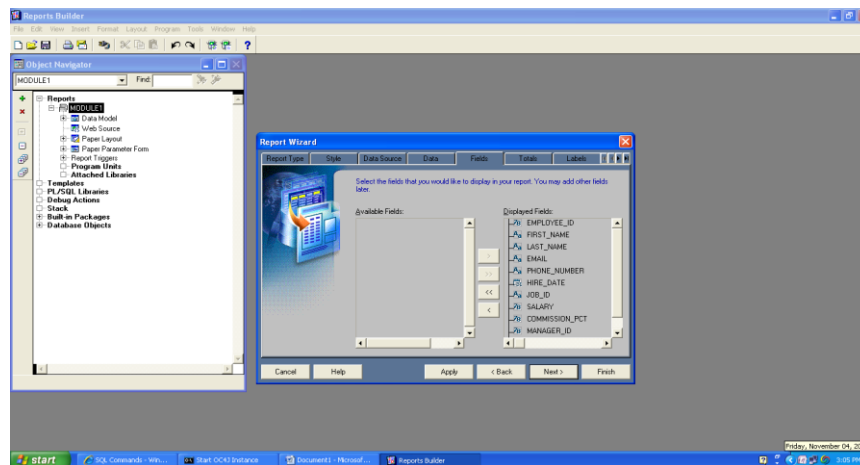
We can alter the sequence of displayed fields by dragging one field above or below another in the list. The sequence of fields in this list determines how the fields appear in the report output. In a tabular report, the fields appear in sequence from left to right across the page.

Fields that remain in the Available Fields list are available for you to reference in your report definition as hidden fields or in PL/SQL trigger code.

In the report output, the user sees only those fields that you transfer to the Displayed Fields list.



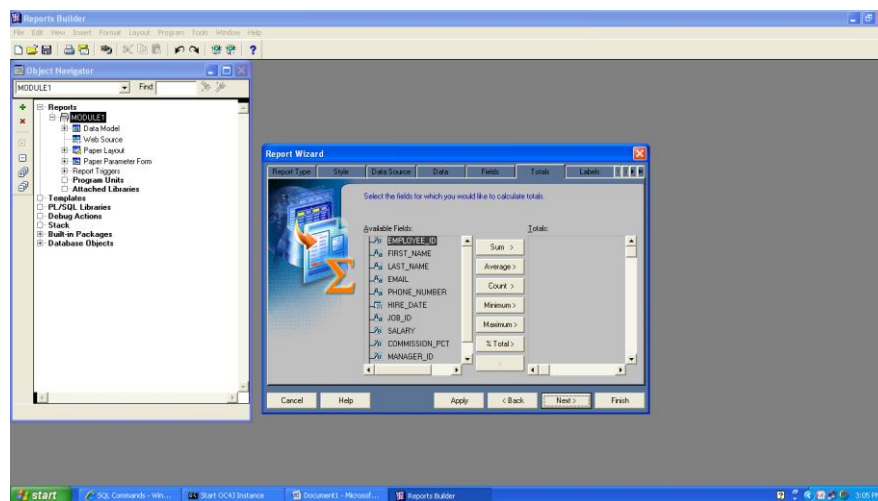




## Step 7: Totals and Labels

In the next two pages of the Report Wizard, you can create totals based on any of the displayed fields and modify the labels and width of the displayed fields.

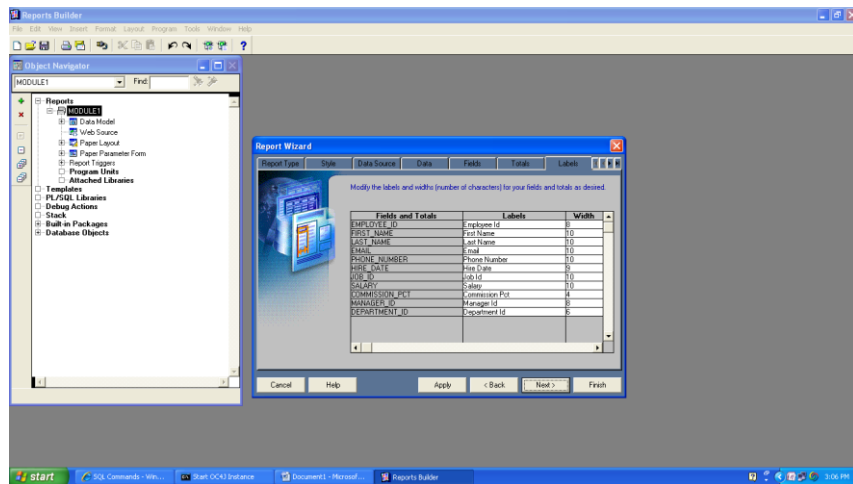
**Totals:** Standard MYSQL aggregate functions are provided for creating totals in your report.



**Labels:** The field label is displayed on one or more lines in the report output. In a tabular report, the labels appear above the field values.

- If the initial label is wider than the field, Reports Builder allows enough space for the label, or displays it on multiple lines.
- If you increase the number of characters in the label text in the reentrant Wizard, the label can appear truncated in the report output.





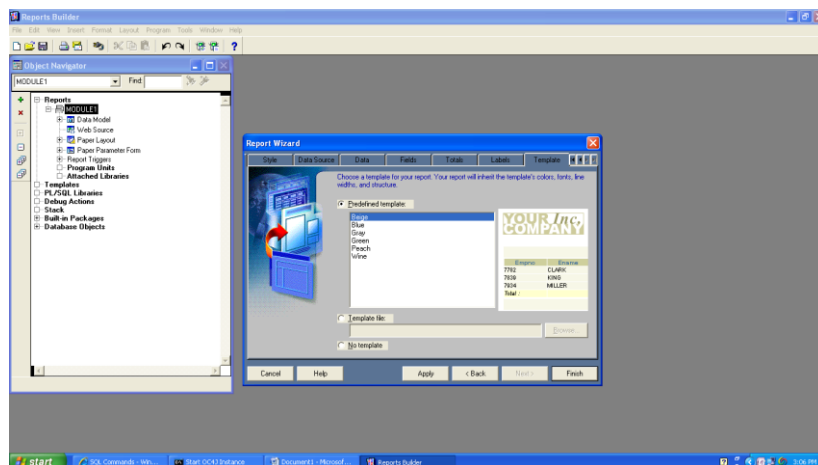
### Step 8: Selecting a Report Template

Report Templates enforce corporate standards as well as create professional-looking paper reports easily.

Select a template from the list of predefined template names. In a template, the fonts, styles, and colors are already selected for designated objects. A variety of templates are available with the standard Reports installation.

To select a predefined template:

1. Select the Predefined Template option button, if it is not already selected.
2. Select a template from the Template list.
3. Click Finish.



### Step 9: Viewing the Paper Report Output

When you finish creating your report in the Report Wizard, the output appears in the Paper Design view of the Report Editor.

#### Magnifying the Output

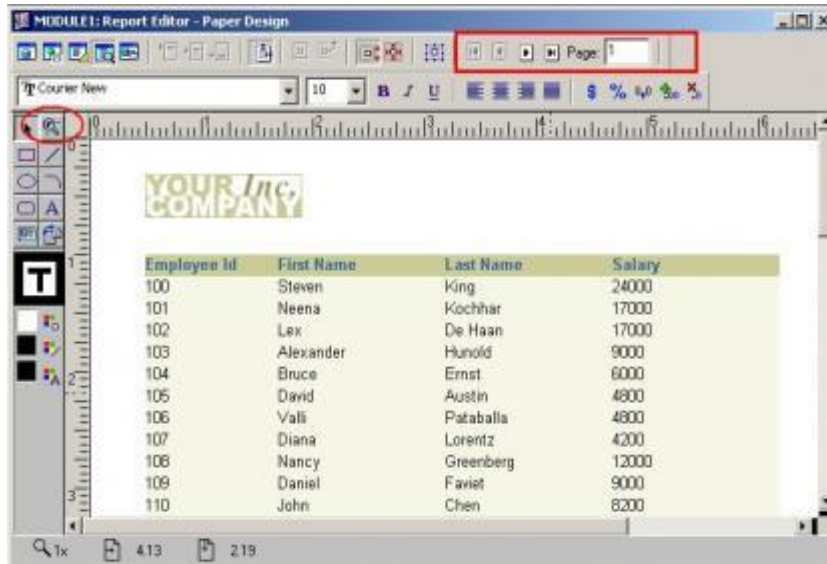
The Paper Design view contains a Magnify tool in the vertical toolbar. This provides a view of the area of layout you want to see. You can also use the View menu to magnify or reduce the size of the output. Select View > Zoom to see your options.

#### Viewing Different Pages



The Paper Design toolbar contains four buttons, and the specific page option, with which you can scroll through the pages of your report.

Report Output



### Step 10: Saving the Report Definition

Remember to save the report frequently by selecting Save in the toolbar, or by using the File > Save menu option. The recommended format for storing paper reports is with an .rdf extension.

If you want to make a copy of the report definition in a different filename, use the menu option File > Save As. There is no toolbar button for the Save As option.



## AIM: CREATE PASSWORDS FOR CREATING APPLICATIONS.

Database security – only authorized users can perform authorized activities

- DBMS products provide security facilities
- They limit certain actions or certain objects to certain users or groups we called as roles
- Almost all DBMS products use some form of username and password security

Principle of least privilege

- Privileges
- A user should have the lowest level of privilege required to perform his assigned task
- GRANT and REVOKE
- GRANT – create users / grant them privileges
- REVOKE – remove privileges
  - Privileges
    - ALL
    - SELECT
    - INSERT, DELETE, UPDATE
    - CREATE, ALTER, DROP

GRANT Syntax GRANT privileges [columns]

ON object

TO user [IDENTIFIED BY 'password']

[WITH GRANT OPTION]

Example:

GRANT ALL

ON dbmusic.\*

TO dbuser IDENTIFIED BY 'userpass'

## REVOKE Syntax

REVOKE priv\_type

ON object

FROM user [, user]

Example:

REVOKE INSERT

ON dbmusic.\*

FROM dbuser

DBMS Security Guidelines Manage accounts and passwords

- ♣ Use a low privilege user account for the DBMS service
- ♣ Protect database accounts with strong passwords
- ♣ Monitor failed login attempts
- ♣ Frequently check group and role memberships
- ♣ Audit accounts with null passwords
- ♣ Limit DBA account privileges

Application Security If DBMS security features are inadequate,

- ♣ additional security code could be written in application program Example?
- ♣ Use the DBMS security features first



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Estd : 2008

Affiliated to Osmania University & Approved by AICTE



To practice basic MYMYSQL DCL Commands such as GRANT, REVOKE, etc.

Syntax:

MYMYSQL>CREATING A USER

MYSQL>Connect system/manager;

MYSQL>Create user "username"identified by "password"

MYSQL>Grant DBA to "username"

MYSQL>Connect "username"/" password";

Example:

MYSQL>Connect 160615733052/123;

MYSQL>Create user cse2 identified by "cse"

MYSQL>Grant DBA to cse2

MYSQL>Connect cse2/cse;

RESULT:

Connected

Syntax

MYSQL>Grant privileges on object\_name to user-name;

Example:

MYSQL>Grant select,update on emp to cse2;

RESULT:

Grant succeeded

Syntax:

MYSQL>Revoke privileges on object\_name from user-name;

Example:

Revoke select,update on emp from cse2;

RESULT:

Revoke succeeded

### **Basic MYSQL TCL Commands.**

To practice basic MYSQL TCL Commands such as COMMIT, SAVEPOINT, ROLLBACK.

Syntax:

MYSQL>Commit;

Example:

MYSQL>Commit;

Result:

Commit completed

Syntax:

MYSQL>Savepoint ID

Example:

MYSQL>Savepoint xyz;



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Estd : 2008

Affiliated to Osmania University & Approved by AICTE

Result:

Savepoint completed



## EXPERIMENT

### AIM: USAGE OF TABLE LOCKING, FACILITIES IN APPLICATIONS.

#### Implementation on Locking

Locking Table.

AIM: To learn commands related to Table Locking

LOCK TABLE Statement Manually lock one or more tables.

Syntax:

```
LOCK TABLE [ schema ..] table[options ] IN lockmode MODE [NOWAIT] LOCK TABLE  
[ schema .] view [ options ] IN lockmode MODE [NOWAIT]
```

Options:PARTITION

( partition)

SUBPARTITION (subpartition)@dblinklockmodes:EXCLUSIVESHAREROW

EXCLUSIVESHARE ROW EXCLUSIVEROW SHARE\* | SHARE UPDATE\*

If NOWAIT is omitted Oracle will wait until the table is available. Several tables can be locked with a single command - separate with commas

e.g. LOCK TABLE table1,table2,table3 IN ROW EXCLUSIVE MODE;

Default Locking Behaviour :

A pure SELECT will not lock any rows.INSERT, UPDATE or DELETE's - will place a ROW EXCLUSIVE lock.SELECT...FROM...FOR UPDATE NOWAIT - will place a ROW EXCLUSIVE loc

#### Client A:

Create table student (sno number(10),sname varchar2(15))

#### Output:

Table created

Insert into student values(12,'sai');

#### Output:

1 row inserted

#### Client B:

Insert into student values(13,'tanvi');

#### Output:

1 row inserted

#### Client A:

Lock table student in exclusive mode nowait;

#### Output:

Table locked

#### Client B:

Insert into student values(14,'pooji');

#### Output:



If u have to insert the values the client b system will be hang, because client A system was locked.

## EXPERIMENT

### AIM: CREATION OF SMALL FLL-FLEDGED OF DATABASE APPLICATION

The Payroll Management System provides the entire tools one may to need to control payroll systems in any organization. The term 'payroll' encompasses every employee of a company who receives a regular wage or other compensation. Some employees may be paid a steady salary while others are paid for hours worked or the number of items produced. All of these different payment methods are calculated by a payroll specialist and the appropriate paychecks are issued. This database gives information about the employee, employer, department in which the employee works and the salary details of the employee.

E-R diagram is the basic step for designing any Database.

This E-R diagram consists of the following entities:

Employee: contains the employee details and has the following attributes

Eid: Employee identity number. It is the primary key.

Sid: Payslip number.

Phno: contact number of the employee.

Ename: contains first name and last name of the Employee.

Cid: company id. It is the foreign key references to cid of Employer entity.

Employer: contains the details of the organization and it has the following attributes.

Company: Name of the organization.

Cid: Company identity. It is the primary key.

Branch: where the company is located

Cno: contains the number of employees working for an organization.

Salary: contains the entire salary details of an employee in an organization.

It has the following attributes.

Sid: Payslip number. It is the primary key.

Basic: Amount of basic salary of the employee.

Paydate: The date when the salary is given to the employee.

Allowances: Amount for allowances.

Deductions: Amount deducted from total for pf, taxes, etc.

Eid: Employee identity. It is the foreign key references to eid of Employee.

Total: The total salary to be paid for the employee. It is derived from basic, allowances and deductions. (basic + allowances – deductions)

Employs: gives the relation between employee and employer and contains the information related to the attendance of the employee. It contains the following descriptive attributes.

Eid: Employee identity. It is the foreign key references to eid of Employee.

Doj: It is the date of joining of the Employee.

Workdays: It is number of working days of the Employee.

Department: gives the information in which department the employee works and the designation of the employee in that department. It contains the following attributes.

Eid: Employee identity. It is the foreign key references to eid of Employee.

Dept: Department in which the employee works.

Designation: Employee designation.

SScale: The salary scale of the Employee.

This database is more compatible and we can apply aggregate functions on this database.

```
create table Employer(cid varchar2(10) primary key,company varchar2(50),branch
varchar2(20),cno number(10))
```



Table created.

```
create table Employee(eid varchar2(10) primary key,sid varchar2(10),firstname
varchar2(15),lastname varchar2(15),phno number(10),cid varchar2(10),foreign key(cid) references
Employer(cid))
```

```
Table created.create table Salary (sid number(10) primary key,eid varchar2(10),basic
number(10),paydate date,allowances number(10),deduction number(10),foreign key(eid) references
Employee(eid))
```

Table created.

```
create table Employs(eid varchar2(10), doj date, workdays number(4),foreign key(eid) references
Employee(eid))
```

Table created.

```
create table Department(eid varchar2(10), dept varchar2(40), designation varchar2(10), SScale
number(10),foreign key(eid) references Employee(eid))
```

Table created.

```
insert into Employer values('L-101','LIC','Hyderabad',100)
1 row(s) inserted.
```

```
insert into Employer values('O-111','OIC','Hyderabad',150)
1 row(s) inserted.
```

```
insert into Employer values('L-102','LIC','Kolkata',200)
1 row(s) inserted.
```

```
insert into Employer values('U-234','UIC','Bombay',400)
1 row(s) inserted.
```

```
insert into Employee values('LH1013007','LH113007','Alpana','Uttara',8142200667,'L-101') 1
row(s) inserted.
```

```
insert into Employee values('LH1013020','LH113020','Divya','Vani',9618132553,'L-101')
1 row(s) inserted.
```

```
insert into Employee values('OH1113005','OH113005','Varsha','Kandi',9700383811,'O-111')
1 row(s) inserted.
```

```
insert into Employee values('UB2343022','UB103022','Harshitha','Reddy',9573041324,'U-234')
1 row(s) inserted.
```

```
insert into Employee values('LK1023011','LK093011','Archana','TR',9030387745,'L-102')
1 row(s) inserted.
```

```
insert into Employee values('OH1113031','OH093031','Keerthi','Ch',9704687420,'O-111')
1 row(s) inserted.
```

```
insert into Employs values('LH1013007','13-AUG-11',24)
1 row(s) inserted.
```

```
insert into Employs values('LH1013020','12-NOV-11',26)
1 row(s) inserted.
```

```
insert into Employs values('OH1113005','12-OCT-11',25)
1 row(s) inserted.
```

```
insert into Employs values('UB2343022','24-JAN-10',26)
```





**METHODIST**  
COLLEGE OF ENGINEERING & TECHNOLOGY  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Estd : 2008

Affiliated to Osmania University & Approved by AICTE



1 row(s) inserted.

insert into Employs values('LK1023011','11-JAN-09',22)

1 row(s) inserted.

insert into Employs values('OH1113031','12-OCT-09',24)

1 row(s) inserted.

insert into Department values('OH1113031','Development','DO',1500)

1 row(s) inserted.

insert into Department values('LK1023011','Advertising','TeamLeader',1800)

1 row(s) inserted.

insert into Department values('UB2343022','Marketing','Officer',1400)

1 row(s) inserted.

insert into Department values('OH1113005','Sales','executive',2000)

1 row(s) inserted.

insert into Department values('LH1013007','Management','Manager',3000)

1 row(s) inserted.

insert into Department values('LH1013020','Administrative','Manager',2200)

1 row(s) inserted.

insert into Salary values('OH093031','OH1113031',1000,'12-NOV-09',2000,500)

1 row(s) inserted.

insert into Salary values('UB103022','UB2343022',1000,'24-FEB-10',3000,800)

1 row(s) inserted.

insert into Salary values('LK093011','LK1023011',1500,'12-NOV-11',2000,500)

1 row(s) inserted.

insert into Salary values('LH113020','LH1013020',1600,'12-NOV-11',2500,700)

1 row(s) inserted.

insert into Salary values('LH113007','LH1013007',1700,'12-NOV-11',2600,900)

1 row(s) inserted.

insert into Salary values('OH113005','OH1113005',1300,'12-NOV-11',2000,700)

1 row(s) inserted.

select \* from Employer

CID	COMPANY	BRANCH	CNO
L-101	LIC	Hyderabad	100
O-111	OIC	Hyderabad	150
L-102	LIC	Kolkata	200
U-234	UIC	Bombay	400



select \* from Employee

EID	SID	FIRST NAME	LAST NAME	PHNO	CID
OH1113005	OH113005	Varsha	Kandi	9700383811	O-111
UB2343022	UB103022	Harshitha	Reddy	9573041324	U-234
LK1023011	LK093011	Archana	TR	9030387745	L-102
OH1113031	OH093031	Keerthi	Ch	9704687420	O-111
LH1013007	LH113007	Alpana	Uttara	8142200667	L-101
LH1013020	LH113020	Divya	Vani	9618132553	L-101

select \* from Employs

EID	DOJ	WORKDAYS
OH1113031	12-OCT-09	24
LK1023011	11-JAN-09	22
LH1013007	13-AUG-11	24
LH1013020	12-NOV-11	26
OH1113005	12-OCT-11	25
UB2343022	24-JAN-10	26

select \* from Department

EID	DEPT	DESIGNATION	SSCALE
OH1113031	Development	DO	1500
LK1023011	Advertising	TeamLeader	1800
UB2343022	Marketing	Officer	1400
OH1113005	Sales	executive	2000
LH1013007	Management	Manager	3000
LH1013020	Administrative	Manager	2200

select \* from Salary

SID	EID	BASIC	PAYDATE	ALLOWANCES	DEDUCTION
OH093031	OH1113031	1000	12-NOV-09	2000	500
OH113005	OH1113005	1300	12-NOV-11	2000	700
LH113007	LH1013007	1700	12-NOV-11	2600	900
LH113020	LH1013020	1600	12-NOV-11	2500	700
LK093011	LK1023011	1500	12-NOV-11	2000	500
UB103022	UB2343022	1000	24-FEB-10	3000	800

## ii) Library management System

In the library Management system, the following entities and attributes can be identified.

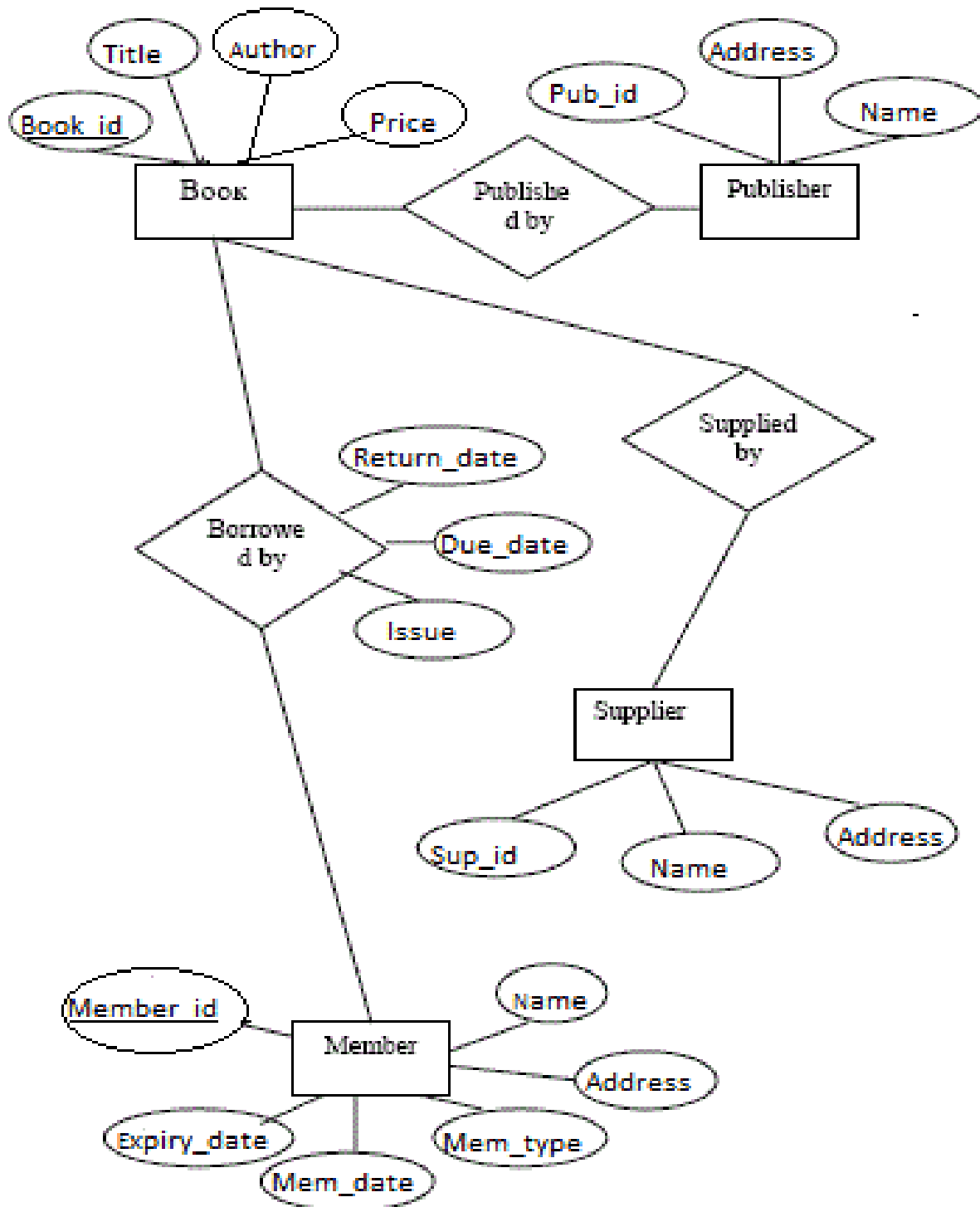
**Assumptions:** A publisher publishes a book. Supplier supplies a book to library. Members borrow the book.

- **Book-** the set of all books in the library.  
**Attributes:** book\_id, title, author, and price.
- **Member-** the set of all library members.  
**Attributes:** member\_id, name, address, mem\_type, mem\_date, expiry\_date.
- **Publisher-** the set of all the publishers of the books.  
**Attributes:** pub\_id, name, address.
- **Supplier-** the set of all the suppliers of the books.  
**Attributes:** sup\_id, name, address.



➤ **Borrowedby:**

Attributes: return\_date, due\_date, issue.





## CREATION OF TABLES:

### Book table:

Create table book(book\_id number(10) primarykey,title varchar2(20), author varchar2(20),price number(20))

Table created.

### Member table:

Create table member(member\_id number(20) primary key, name varchar2(20), address varchar2(20), mem\_type varchar2(20), mem\_date date, expiry\_date date)

Table created.

### Publisher table:

Create table publisher(pub\_id number(20), name varchar2(20), address varchar2(20))

Table created.

### Supplier table:

Create table supplier(sup\_id number(20), name varchar2(20), address(20))

Table created.

### Borrowedby table:

Create table borrowedby(return\_date date, due\_date date, issue date)

Table created.

## INSERTING VALUES INTO TABLES:

### Book table:

Insert into book values(3035,'DBMS','Forth',645)

1 row(s) inserted.

Insert into book values(3050,'CN','Tanenbaurn',455)

1 row(s) inserted.

Select \* from book

BOOK_ID	TITLE	AUTHOR	PRICE
3035	DBMS	Forth	645
3050	CN	Tanenbaurn	455

### Member table:

Insert into member values(3040,'padma','nagole','b',16/11/2011,23/11/2011)

1 row(s) inserted.

Insert into member values(3045,'latha','abids','a',18/11/2011,25/11/2011)

1 row(s) inserted.

Select \* from member

MEMBER_ID	NAME	ADDRESS	MEM_TYPE	MEM_DATE	EXPIRY_DATE
3040	Padma	Nagole	B	16/11/2011	23/11/2011
3045	Latha	abids	A	18/11/2011	25/11/2011

### Publisher table:

Insert into publisher values (8050,'sudarshan','bombay')

1 row(s) inserted.

Insert into publisher values(8035,'harihara','orissa')

1 row(s) inserted.

Select \* from publisher

PUB_ID	NAME	ADDRESS
8050	sudarshan	Bombay
8035	Harihara	Orissa



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Estd : 2008

Affiliated to Osmania University & Approved by AICTE



### Supplier table:

Insert into supplier values (3535,'sachin','guntur')

1 row(s) inserted.

Insert into supplier values(5050,'pradyumna','nijamabad')

1 row(s) inserted.

Select \* from supplier

SUP_ID	NAME	ADDRESS
3535	sachin	Guntur
5050	Pradyumna	nijamabad

### Borrowedby table:

Insert into borrowedby values (20/11/2011,23/11/2011,16/11/1011)

1 row(s) inserted.

Insert into borrowedby values (25/11/2011,25/11/2011,18/11/2011)

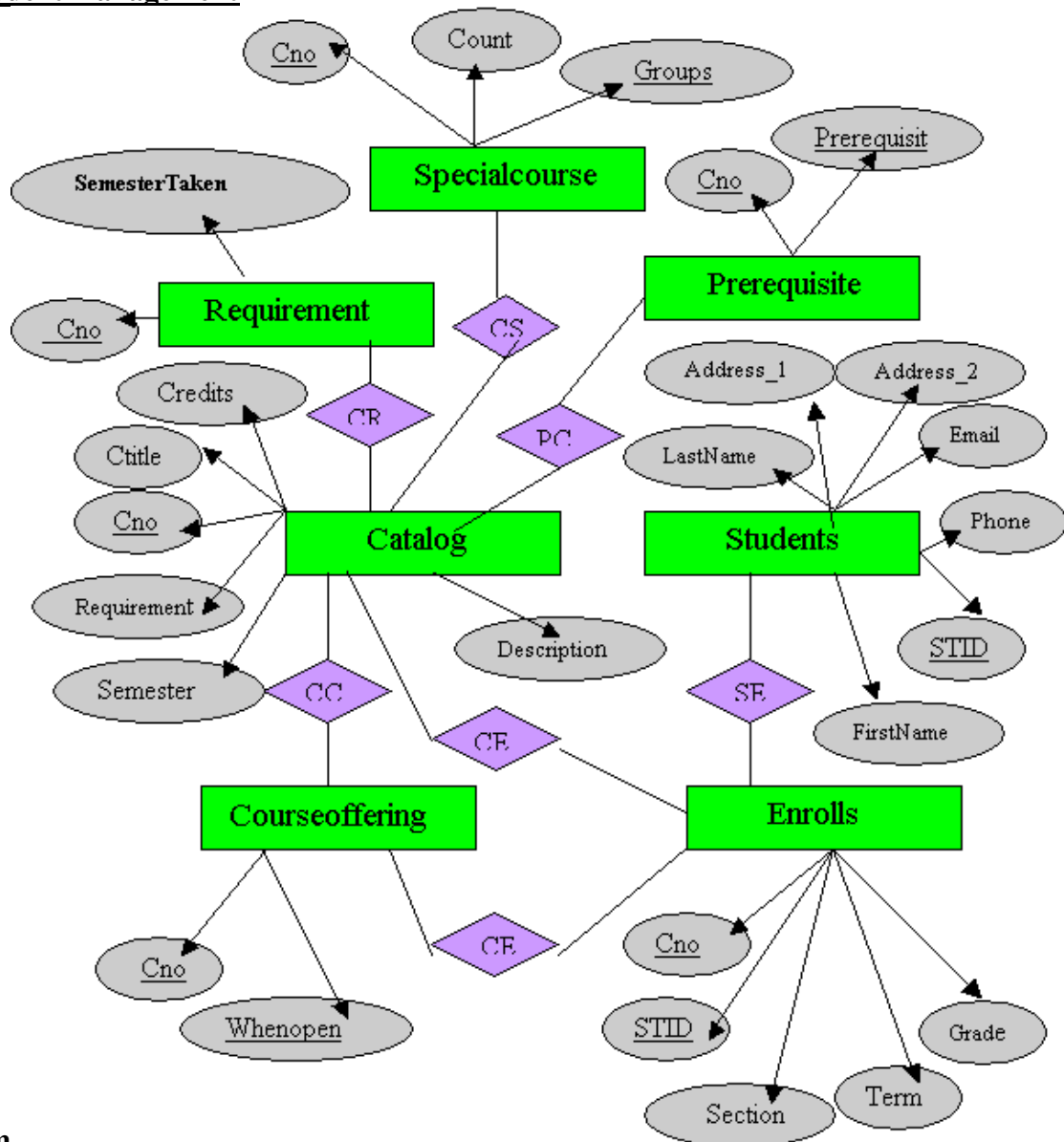
1 row(s) inserted.

Select \* from borrowedby

RETURN_DATE	DUE_DATE	ISSUE
20/11/2011	23/11/2011	16/11/2011
25/11/2011	25/11/2011	18/11/2011



**iii) Student management**



**System**

Catalog3 Table

Create table catalog3(Cno varchar2(12) primary key,Ctitle varchar2(50) null,Credits number(2) null,Semester varchar2(12) null,Requirement varchar2(11) null,Description varchar2(600) null)

Table created.

Students Table

Create table students (STID varchar2(11) primary key,FirstName varchar2(20) null,LastName varchar2(20) null,Address\_1 varchar2(30) null,Email varchar2(50) null,Phone varchar2(15) null)

Table created

Enrolls Table

Create table enrolls (STID varchar2(11) not null,Cno varchar2(12) not null,Term varchar2(20) not null,Section varchar2(15) not null,Grade varchar2(6) null,Primary key (Cno, STID),Foreign key (STID) references students,Foreign key (Cno) references catalog3)

Table created

Prerequisite Table

Create table prerequisite (Cno varchar2(12) not null,Prerequisite varchar2(12) null,Primary key (Cno, prerequisite),Foreign key (Cno) references catalog3)

Table created

Courseoffering Table

Create table courseoffering (Cno varchar2(12) not null,WhenOpen varchar2(12) not null,Primary key (Cno, WhenOpen),Foreign key (Cno) references catalog3)

Table created

Specialcourse Table

Create table specialcourse (Cno varchar2(12) not null,Groups varchar2(60) not null,Counts number(2) null,Primary key (Cno, Groups),Foreign key (Cno) references catalog3)

Table created

Requirement Table

Create table requirement (Cno varchar2(12) not null,SemesterTaken number(14) not null,Primary key (Cno, SemesterTaken),Foreign key (Cno) references Catalog3)

Table created

Insertions into the tables

insert into catalog3 values('CS102','Cmputg',3,'Sem3','program','Computing')

1 row (s) inserted

insert into catalog3 values('AD101','FARts',3,'Sem6','general','valarts')

1 row (s) inserted

select \* from catalog3

CNO CTITLE CREDTS SEMESTER REQUIREMENT DESCRIPTION

CS102 Cmputg 3 Sem3 program Computing

AD101 FARts 3 Sem6 general valarts

insert into students values ('98','yineli','Zheng','CaptialWa','[zheng@yahoo.com](mailto:zheng@yahoo.com)','8756')

1 row (s) inserted

insert into students values ('22','Xiaohng','Qian','CindyDr','[yunlei@yahoo.com](mailto:yunlei@yahoo.com)','1704')

1 row (s) inserted

select \* from students

STID FIRSTNAME LASTNAME ADRES\_1 EMAIL PHONE

98 yineli Zheng CaptialWa [zheng@yahoo.com](mailto:zheng@yahoo.com) 8756

22 Xiaohng Qian CindyDr [yunlei@yahoo.com](mailto:yunlei@yahoo.com) 1704

insert into enrolls values ('98', 'CS102','sem2','SPring99','B+')

1 row (s) inserted

insert into enrolls values('22','AD101','Sem3','SPring66','A+')

1 row (s) inserted

select \* from enrolls





STID CNO TERM SECTION GRADE

98 CS102 Sem2 SPring99 B+

22 AD101 Sem3 SPring66 A+

insert into prerequisite values('AD101','CPE286')

1 row (s) inserted

insert into prerequisite values('CS102','CPE210')

1 row (s) inserted

select \* from prerequisite

CNO PREREQUISITE

AD101 CPE286

CS102 CPE210

insert into requirement values('AD101','5')

1 row (s) inserted

insert into requirement values('CS102','3')

1 row (s) inserted

select \* from requirement

CNO SEMESTERTAKEN

AD101 5

CS102 3

insert into specialcourse values('AD101','G2','1')

1 row (s) inserted

insert into specialcourse values('CS102','G5','6')

1 row (s) inserted

select \* from specialcourse

CNO GROUPS COUNTS

AD101 G2 1

CS102 G5 6

insert into courseoffering values('CS102','Spring')

1 row (s) inserted

insert into courseoffering values('AD101','Fall')

1 row (s) inserted

select \* from courseoffering

CNO WHENOPEN

CS102 Spring

AD101 Fall

Queries

calculate the average of credits

select avg(credits) from catalog3

AVG(CREDITS)

3

display students firstname whose second letter is 'i'

select firstname from students where firstname like '\_i%'

FIRSTNAME

yneli

Xiaohong

display students lastname whose grade is 'B+'

select lastname from students where stid in(select stid from enrolls where grade='B+')

LASTNAME

Zheng



display the semester taken by the student whose firstname is 'yinelì'

select semestertaken from requirement where cno in (select cno from catalog where cno in (select cno from enrolls where stid in (select stid from students where firstname='yinelì'))))

SEMESTERTAKEN

3

display the student id's who are enrolled

select stid from students intersect select stid from enrolls

STID

22

98



## ADDITIONAL EXPERIMENT 1

### AIM: EXERCISING STORED FUNCTIONS

#### **MYSQL Aggregate Functions**

MYSQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- **AVG ()** - Returns the average value
- **COUNT ()** - Returns the number of rows
- **MAX ()** - Returns the largest value
- **MIN ()** - Returns the smallest value
- **SUM ()** - Returns the sum

#### **1. The AVG () Function**

The AVG () function returns the average value of a numeric column.

##### **MYSQL AVG ()**

###### **Syntax**

```
SELECT AVG (column_name) FROM table_name
```

#### **2. The COUNT () Function**

The COUNT () function returns the number of rows that matches a specified criterion.

##### **MYSQL COUNT(column\_name)**

###### **Syntax**

The COUNT(column\_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name
```

##### **MYSQL COUNT (\*)**

###### **Syntax**

The COUNT (\*) function returns the number of records in a table:

```
SELECT COUNT (*) FROM table_name
```

##### **MYSQL COUNT (DISTINCT column\_name)**

###### **Syntax**

The COUNT (DISTINCT column\_name) function returns the number of distinct values of the specified column:

```
SELECT COUNT (DISTINCT column_name) FROM table_name
```

#### **3. The MAX () Function**

The MAX () function returns the largest value of the selected column.

##### **MYSQL MAX ()**

###### **Syntax**

```
SELECT MAX(column_name) FROM table_name
```

#### **4. The MIN() Function**

The MIN() function returns the smallest value of the selected column.



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

**Accredited by NAAC with A+ and NBA**

Estd : 2008

Affiliated to Osmania University & Approved by AICTE



**MYSQL MIN()**

**Syntax**

SELECT MIN(column\_name) FROM table\_name



### 5. The SUM () Function

The SUM () function returns the total sum of a numeric column.

#### MYSQL SUM ()

##### Syntax

SELECT SUM(column\_name) FROM table\_name

List minimum, maximum, average & sum of salaries of employee.

#### EXAMPLE

MYSQL>select min(sal), max(sal),avg(sal),sum(sal) from emp;

#### RESULT:

<u>MIN(SAL)</u>	<u>MAX(SAL)</u>	<u>AVG(SAL)</u>	<u>SUM(SAL)</u>
1250	2975	2168.75	8675

#### MYSQL Date Data Types

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MM:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MM:SS
- YEAR - format YYYY or YY

In Oracle/PLMYSQL, the **to\_char** function converts a number or date to a string.

The syntax for the **to\_char** function is:

to\_char( value, [ format\_mask ], [ nls\_language ] )

value can either be a number or date that will be converted to a string.

format\_mask is optional. This is the format that will be used to convert value to a string.

nls\_language is optional. This is the nls language used to convert value to a string.

Applies To:

- Oracle 8i, Oracle 9i, Oracle 10g, Oracle 11g

#### Examples:

<u>FirstName</u>	<u>LastName</u>	<u>DateOfBirth</u>	<u>Email</u>	<u>City</u>
John	Smith	12/12/1969	john.smith@john-here.com	New York
David	Stonewall	01/03/1954	david@MYSQL-tutorial.com	San Francisco
Susan	Grant	03/03/1970	susan.grant@MYSQL-tutorial.com	Los Angeles
Paul	O'Neil	09/17/1982	paul.oneil@pauls-email.com	New York
Stephen	Grant	03/03/1974	sgrant@sgrantemail.com	Los Angeles

SELECT FirstName, LastName FROM Users WHERE DateOfBirth > '02/03/1970'

>= (Greater or Equal)

SELECT FirstName, LastName FROM Users WHERE DateOfBirth >= '02/03/1970'

< (Less than)





```
SELECT FirstName, LastName FROM Users WHERE DateOfBirth < '02/03/1970'
<= (Less or Equal)
SELECT FirstName, LastName FROM Users WHERE DateOfBirth <= '02/03/1970'
```

In addition to the comparison operators you can use WHERE along with logical operators. MySQL logical Operators are used to combine two or more criterions in the WHERE clause of an MYSQL statement. If we want to select all users from our Users table, which live in New York and are born after 10/10/1975

We will use the following MYSQL query:

```
SELECT FirstName, LastName, DateOfBirth, Email, City FROM Users WHERE City = 'New
York' AND DateOfBirth > '10/10/1975'
```

Here is the result of the above SELECT:

<u>FirstName</u>	<u>LastName</u>	<u>DateOfBirth</u>	<u>Email</u>	<u>City</u>
Paul	O'Neil	09/17/1982	paul.oneil@pauls-email.com	New York

As you can see we now have to criteria concatenated with the AND logical operator, which means that both conditions have to be true. If we want to select all users from our Users table, which live in New York or are born after 10/10/1975 we will use the following MYSQL query:

```
SELECT FirstName, LastName, DateOfBirth, Email, City FROM Users WHERE City = 'New
York' OR DateOfBirth > '10/10/1975'
```

The result is:

<u>FirstName</u>	<u>LastName</u>	<u>DateOfBirth</u>	<u>Email</u>	<u>City</u>
John	Smith	12/12/1969	john.smith@john-here.com	New York
Paul	O'Neil	09/17/1982	paul.oneil@pauls-email.com	New York
Stephen	Grant	03/03/1974	sgrant@sgrantemail.com	Los Angeles

### **The Built-In Date Functions**

#### **Name Description**

**ADD\_MONTHS** Adds the specified number of months to a date.

Example: ADD\_MONTHS ('31-JAN-1995', 1) ==> 28-FEB-1995

**LAST\_DAY** Returns the last day in the month of the specified date.

#### **Example:**

Go to the last day in the month:

LAST\_DAY ('12-JAN-99') ==> 31-JAN-1999

**MONTHS\_BETWEEN** Calculates the number of months between two dates.

Example: MONTHS\_BETWEEN ('31-MAR-1995', '28-FEB-1994') ==> 13

**NEXT\_DAY** Returns the date of the first weekday specified that is later than the date.

Example: NEXT\_DAY ('01-JAN-1997', 'monday') ==> 06-JAN-1997



**ROUND** Returns the date rounded by the specified format unit.

**Example:**

Round down and up to the first of the year:

ROUND (TO\_DATE ('01-MAR-1994'), 'YYYY') ==> 01-JAN-1994

ROUND (TO\_DATE ('01-SEP-1994'), 'YEAR') ==> 01-JAN-1995

**SYSDATE** Returns the current date and time in the Oracle Server.

**Example:** Select sysdate from dual

**TRUNC**

**Truncates the specified date of its time portion according to the format**

Trunc to the first of the current year:

TRUNC (TO\_DATE ('01-MAR-1994'), 'YYYY') ==> 01-JAN-1994

TRUNC (TO\_DATE ('01-SEP-1994'), 'YEAR') ==> 01-JAN-1994

Trunc back to the beginning of the current day (time is always midnight):

TO\_CHAR (TRUNC (TO\_DATE ('11-SEP-1994 10:00 AM',

'DD-MON-YYYY HH:MI AM'), 'DD'),

'DD-MON-YYYY HH:MI AM')

==> 11-SEP-1994 12:00 AM

TO\_CHAR (TRUNC (TO\_DATE ('11-SEP-1994 4:00 PM',

'DD-MON-YYYY HH:MI AM'), 'DD'),

'DD-MON-YYYY HH:MI AM')

==> 11-SEP-1994 12:00 AM

**Other Examples:**

Day of first of year is Saturday:

TO\_CHAR (TO\_DATE ('01-JAN-1994'), 'DAY') ==> 'SATURDAY'

First day in the month is a Friday:

TO\_CHAR (TO\_DATE ('01-APR-1994'), 'DAY') ==> FRIDAY

**Example :**

Assume we have the following "Orders" table:

<u>OrderId</u>	<u>ProductName</u>	<u>OrderDate</u>
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Now we want to select the records with an OrderDate of "2008-11-11" from the table above.

The result-set will look like this:

<u>OrderId</u>	<u>ProductName</u>	<u>OrderDate</u>
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11



Now, assume that the "Orders" table looks like this (notice the time component in the "OrderDate" column):

<u>OrderId</u>	<u>ProductName</u>	<u>OrderDate</u>
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

If we use the same SELECT statement as above:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

we will get no result! This is because the query is looking only for dates with no time portion

## AIM : PL SQL FUNCTIONS

A function is same as a procedure except that it returns a value.

### 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 the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a **return** statement.
- The RETURN clause specifies the 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.

### Function (Factorial of N numbers)

```
MYSQL> create or replace function fac(n integer)
```

```
return integer is
```

```
    f number;
```

```
    begin
```

```
f:=1;
```

```
    for i in 1..n
```

```
    loop
```

```
f:=f*i;
```

```
    end loop;
```

```
    return f;
```



end;

/

Function created.

MYSQL> select fac(4) from dual;

FAC (4)

-----  
24

```
Run SQL Command Line

SQL> select * from depositor;

CUSTOMERNAME      ACCOUNTNUMBER      BALANCE  BRANCHNAME
-----
sharon             123345             10000    tarnaka
rithika            23456              15000    secunderabad
ruhi               45678              10000    abids
sharon            123345             10000    tarnaka

SQL> create or replace function totbal
2  return number IS
3  total number(10):=0;
4  begin
5  select count(*) into total from depositor;
6  return total;
7  end;
8  /

Function created.

SQL> declare
2  c number(2);
3  begin
4  c:=totbal();
5  dbms_output.put_line('Total no of cust'||c);
6  end;
7  /

Total no of cust4

PL/SQL procedure successfully completed.

SQL> _
```



The screenshot shows a Windows desktop with a taskbar at the bottom. There are two Notepad windows and one SQL Command Line window. The top Notepad window, titled 'totbal1.sql - Notepad', contains the following SQL code:

```
create or replace function totbal1
return number is
total number(10):=0;
begin
select count(*) into total from depositor;
return total;
end;
/
```

The bottom Notepad window, titled 'calctotbal1.sql - Notepad', contains the following SQL code:

```
declare
c number(2);
begin
c:=totbal1();
dbms_output.put_line('Total'||c);
end;
```

The SQL Command Line window, titled 'Run SQL Command Line', shows the execution of the first script:

```
SQL> @C:\users\hys1\desktop\vai\totbal1.sql;
Function created.
SQL> @C:\users\hys1\desktop\vai\calctotbal1.sql;
7 /
Total4
PL/SQL procedure successfully completed.
SQL>
```

```
create or replace function fac(n integer)
return integer is
f number;
begin
f:=1;
for i in 1..n
loop
f:=f*i;
end loop;
return f;
end;
/
```

The screenshot shows the SQL Command Line window with the following output:

```
SQL> @C:\users\hys1\desktop\vai\fac.sql
Function created.
SQL> select fac(4) from dual;

      FAC(4)
-----
          24

SQL>
SQL>
SQL>
```



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

Accredited by NAAC with A+ and NBA

Estd : 2008

Affiliated to Osmania University & Approved by AICTE



```
create or replace function is_digit(inchar varchar2)
return boolean is
begin
if(substr(inchar,1,1)in('0','1','2','3','4','5','6','7','8','9'))
then
return true;
else
return false;
end if;
exception
when others then
return false;
end;
call isdigit:
declare
ch char:=&in;
begin
if is_digit(ch)=true then
dbms_output.put_line('is a character');
end if;
end;
```



```

-- isdigit.sql
create or replace function is_digit(inchar varchar2)
return boolean is
begin
if(substr(inchar,1,1)in
('0','1','2','3','4','5','6','7','8','9'))
then
return true;
else
return false;
end if;
exception
when others then
return false;
end;

-- calisdigit.sql
declare
ch char:=&in;
begin
if is_digit(ch)=true then
dbms_output.put_line('is a character');
end if;
end;

-- Run SQL Command Line
SQL>
SQL>
SQL> @C:\users\hys1\desktop\vai\isdigit.sql
14 /

Function created.

SQL> @C:\users\hys1\desktop\vai\calisdigit.sql
8 /
Enter value for in: 2
old 2: ch char:=&in;
new 2: ch char:=2;
is a character

PL/SQL procedure successfully completed.
SQL>

```

```

create or replace function large23(a in number,b in number) return number is
begin
if a>b then
    return a;
else
    return b;
end if;
end;
Function created

```

```

MYSQL>select large23(50,12 from dual;
LARGE23(50,12)
50

```

```

-- large23.sql
create or replace function large23(a in
number,b in number) return number is
begin
if a>b then
    return a;
else
    return b;
end if;
end;

-- Run SQL Command Line
SQL> @C:\users\hys1\desktop\vai\large23.sql
10 /

Function created.

SQL> select large23(50,12) from dual;

LARGE23(50,12)
-----
50

SQL>

```



## ADDITIONAL EXPERIMENT

### AIM: IMPLICIT CURSOR AND EXPLICIT CURSOR

Oracle creates a memory area, known as the context area, for processing an MYSQL statement, which contains all the information needed for processing the statement; for example, the number of rows processed, etc.

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

We can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the MYSQL statement, one at a time. There are two types of cursors:

- Implicit cursors
- Explicit cursors

#### Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an MYSQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **MYSQL cursor**, which always has attributes such as **%FOUND**, **%ISOPEN**, **%NOTFOUND**, and **%ROWCOUNT**. The MYSQL cursor has additional attributes, **%BULK\_ROWCOUNT** and **%BULK\_EXCEPTIONS**, designed for use with the **FORALL** statement. The following table provides the description of the most used attributes:

Any MYSQL cursor attribute will be accessed as **MYSQL%attribute\_name**.

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

Attribute	Description
<b>%FOUND</b>	Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
<b>%NOTFOUND</b>	The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
<b>%ISOPEN</b>	Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
<b>%ROWCOUNT</b>	Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.



The syntax for creating an explicit cursor is:

CURSOR cursor\_name IS select\_statement;

Working with an explicit cursor includes the following steps:

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Create a table following structure.(table name as 'emp')

ename	varchar2(20)
eno	number(10)
doj	date
sal	number(6)
comm	number(5)
deptno	number(10)

```
MYSQL> create table emp(ename varchar2(20),eno number(10),doj date,sal number(6),
comm number(5),deptno number(10));
```

Table created.

```
MYSQL> desc emp;
```

Name	Null?	Type
ENAME		VARCHAR2(20)
ENO		NUMBER(10)
DOJ		DATE
SAL		NUMBER(6)
COMM		NUMBER(5)
DEPTNO		NUMBER(10)

```
MYSQL> insert into emp values('Arihanth',10,'01-jan-06',20000,4000,3);
1 row created.
```

```
MYSQL> insert into emp values('Babu',11,'04-feb-06',15000,3500,2);
1 row created.
```

```
MYSQL> insert into emp values('Hari',13,'05-apr-06',25000,5500,1);
1 row created.
```

```
MYSQL> select * from emp;
```

ENAME	ENO	DOJ	SAL	COMM	DEPTNO
Arihanth	10	01-JAN-06	20000	4000	3
Babu	11	04-FEB-06	15000	3500	2
Hari	13	05-APR-06	25000	5500	1

\*Create a table following structure (table name as 'cust')

cname	varchar2(20)
cno	number(10)



MYSQL> create table cust(cname varchar2(20),cno number(10));

Table created.

MYSQL> desc cust;

Name	Null?	Type
CNAME		VARCHAR2(20)
CNO		NUMBER (10)

MYSQL> insert into cust values('Mani',5);

1 row created.

MYSQL> insert into cust values('Anand',6);

1 row created.

MYSQL> insert into cust values('Arun',8);

1 row created.

MYSQL> insert into cust values('Raj',800);

1 row created.

MYSQL> select \* from cust;

CNAME	CNO
Mani	5
Anand	6
Arun	8
Raj	800

**Write a PL/SQL block to change the date format using implicit cursor.**

MYSQL> Begin

```

update emp set doj = Sysdate where eno=10;
if MYSQL % notfound then
insert into emp values('Rahul',10,'14-may-06',5000,500,3);
end if;
end;
/

```

PL/SQL procedure successfully completed.

MYSQL> set serveroutput on;

MYSQL> select \* from emp;

ENAME	ENO	DOJ	SAL	COMM	DEPTNO
Arihanth	10	24-JUL-07	20000	4000	3
Babu	11	04-FEB-06	15000	3500	2
Hari	13	05-APR-06	25000	5500	1

**2. Write a PL/SQL block to change employee number 10 to employee number 1 using implicit cursor.**

MYSQL> Declare

```

empno number:=1;
begin
update emp set eno=empno where eno=10;
if MYSQL % notfound then
DBMS_OUTPUT.PUT_LINE('Error');
else
DBMS_OUTPUT.PUT_LINE ('Successfully updated');

```



end if;

end; /

PL/SQL procedure successfully completed.

MYSQL> select \* from emp;

ENAME	ENO	DOJ	SAL	COMM	DEPTNO
Arihanth	1	24-JUL-07	20000	4000	3
Babu	11	04-FEB-06	15000	3500	2
Hari	13	05-APR-06	25000	5500	1

**Write a PL/SQL to display customer name for a given customer number using implicit cursor.**

MYSQL>

Declare

cust\_name cust.cname % type;

begin

select cname into cust\_name from cust where cno=800;

if MYSQL%rowcount=1 then

DBMS\_OUTPUT.PUT\_LINE (cust\_name||' has id 800');

end if;

EXCEPTION

when no\_data\_found then

DBMS\_OUTPUT.PUT\_LINE ('customer 800 is not found in the database');

end;

/

Raj has id 800

**Write a PL/SQL block to display records in customer table using explicit cursor.**

MYSQL> declare

cursor c\_cust is select \* from cust;

v\_cust c\_cust % rowtype;

begin

open c\_cust;

loop

fetch c\_cust into v\_cust;

exit when c\_cust % notfound;

DBMS\_OUTPUT.PUT\_LINE (v\_cust.cname||' '||v\_cust.cno);

end loop;

close c\_cust;

end;

/

Mani 5

Anand 6

Arun 8

Raj 800

**Write a PL/SQL to display customer names having customer number less than 700 using explicit cursor.**

MYSQL> declare

cursor c\_cust is select cname from cust where cno<700;

v\_cust c\_cust%rowtype;



**METHODIST**  
**COLLEGE OF ENGINEERING & TECHNOLOGY**  
(An UGC-AUTONOMOUS INSTITUTION)

**Accredited by NAAC with A+ and NBA**

**Affiliated to Osmania University & Approved by AICTE**



Estd : 2008

begin

open c\_cust;

loop fetch c\_cust into v\_cust;

exit when c\_cust%notfound;

DBMS\_OUTPUT.PUT\_LINE(v\_cust.cname);

end loop;

close c\_cust;

end;

/

Mani

Anand

Arun

PL/SQL procedure successfully completed.