University of Pune

Relational Database Management System (MySQL) (CS-212)

S.Y.B.Sc.(Computer Science)

Semester I

Advisors:
Prof. A. G. Gangarde (Chairman, BOS-Comp. Sc.)

Chairman:
Prof. S. S. Deshmukh (Head & Vice-Principal, Modern College, Pune 5)

Co-ordinator:
Mr. A. V. Sathe

Authors:
Mrs. Vaishali Bhoite (Modern College, Shivajinagar, Pune - 5)
Mrs. Shilpa Dange (Modern College, Shivajinagar, Pune - 5)
Mrs. Sampada Vidhvans (Abasaheb Garware College, Pune -5)
Mrs. Jyoti Yadav (B. R. Gholap College, Sangavi , Pune - 27)
Mrs. Kalpana Joshi (Fergusson College , Pune)

Table of Contents

(Guidelines :
Preliminary exercise and Exercise No. 1 are compulsory exercises with no weightage of marks. 5 Marks each for Exercises 2,3,4,5,6)

About the Work Book

Objectives of this book

This workbook is intended to be used by S.Y.B.Sc(Computer Science) students for the two computer science laboratory courses.
The objectives of this book are
1. The scope of the course.
2. Bringing uniformity in the way the course is conducted across different colleges.
3. Continuous & uniform assessment of the students.
4. Providing ready references for students while working in the lab.

How to use this book?

This book is mandatory for the completion of the laboratory course. It is a measure of the performance of the students in the laboratory for the entire duration of the course.

Instructions to the students

1) Students should carry this workbook during practical sessions of computer science.
2) Students should maintain separate journal for the source code, SQL queries/commands along with outputs.
3) Student should read the topics mentioned in Reading section of this book before coming for practical.
4) Students should solve only those exercises which are selected by practical in-charge as a part of journal activity. However, students are free to solve additional exercises to do more practice for their practical examination.

Exercise Set	Difficulty Level	Rule
Self Activity	-----------	Student should solve these exercises for practice only.
SET A	Easy	All exercises are compulsory in this set.
SET B	Medium	At least two exercises are compulsory in this set.
SET C	Difficult	At least one exercise is compulsory in this test.

5) Students will be assessed for each exercise on a scale of 5

1. Note Done	0
2. Incomplete	1
3.Late Complete	2
4.Needs Improvement	3
5.Complete	4
6.Well Done	5

Instructions to the Practical In-charge

1) Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software.
2) Choose appropriate problems to be solved by students.
3) After a student completes a specific set, the instructor has to verify the outputs and sign in the provided space after the activity.
4) Ensure that the students use good programming practices.
5) You should evaluate each assignment carried out by a student on a scale of 5 as specified above ticking appropriate box.
6) The value should also be entered on assignment completion page of respected lab course.

Instructions to the Lab Administrator

1) MySQL 5.0.10 or Greater Version for installation is advised to support PL/SQL and more facilities.
2) MySQL package can be downloaded from the Website http://dev.mysql.com

Installation and Configuration of MySQL on Linux

1. If MySQL service is already installed in Linux , start the service by giving the command

```
<system prompt>   service mysqld start
```

To stop the service , give the command ,

```
<system prompt>   service mysqld stop
```

2 Installing MySQL on linux Using RPMs

I) Login as root . Create a subdirectory under the root directory (/) and copy the RPMs to the subdirectory e.g. /mysqlrpms

```
<system prompt> cd /
<system prompt> mkdir mysqlrpms
<system prompt> cd mysqlrpms
```

II) Install the RPMs by giving following

```
   mount   /media/cdrecorder
cp /media/cdrecorder/Setup/MySQL_Setup/Linux/*.*   /mysqlrpms/
```

III) Change the directory to mysqlrpms

```
mysqlrpms]  rpm -ivh *.rpm
```

Issue the following commands in /usr/bin directory

```
 /usr/bin]  mysql_install_db
 /usr/bin] mysqld_safe
```

IV) Check whether /etc/my.cnf exists ,if not , copy it from the path /usr/share/mysql .

Make the necessary changes in /etc/my.cnf file to support the foreign key constraint . (my.cnf file contents are different from version to version)

Add the following line in my.cnf file at appropriate place (after the line for BDB tables)

```
default-storage-engine=INNODB
```

V) Start the mysql service

Preliminary Exercise

start date : ___/____/

Objectives

To learn about:

How to create Database Pool?

Use the same database pool in all the forthcoming assignments.

Reading

You should read the following topics before starting the exercise

1. All DDL and DML statements.
2. Normal Forms: 1NF, 2NF,3NF.

Ready References

MYSQL is a popular Relational Database Management System (RDBMS) from the open source domain. It is constantly undergoing change and evolving.

MYSQL has built in support for multiple data storage engines that handle different table types.The storage engines are ISAM , MyISAM , Heap , Merge , InnoDB , BDB etc. InnoDB provides MySQL with a transaction-safe data storage engine with commit , rollback and crash recovery capabilities. InnoDB supports transactions, row-level locking, and foreign keys . In Linux InnoDB should be the default database . You can see the default storage engine by issuing the command SHOW ENGINES ; at mysql prompt . If it is other than InnoDB, this can be changed by adding or updating default storage engine specification in the my.cnf file available under /etc/my.cnf .

MYSQL 5.0.10 OR GREATER VERSION INSTALLATION IS ADVICED TO SUPPORT PLSQL AND MORE FACILITIES.

1. Starting MySQL from system prompt:

```
<system prompt>mysql -h <hostname> -u <username> -p
 Enter password:********
```

2. Issue the following commands:

To create a new database :

```
mysql> create database test ;
Here database test is created.
```

3. To get the list of previously created databases:

```
mysql> show databases;
```

4. To connect to the particular database:

```
mysql> use  <database_name>;
```

5. To create a table e.g. T1:

```
mysql>create table t1 ( a int , b float(5,1)) engine =
InnoDB;
Query OK, 0 rows affected (0.60 sec)
```

engine = InnoDB is optional . If default engine is InnoDB , issue create table command without specifying engine.

6. If field a of t1 has to be primary key then give:

```
mysql> create table t1 ( a int primary key , b
float(5,1)) ;
```

If primary key is not specified at the time of table creation, add primary key constraint by giving:

```
mysql> alter table t1 add constraint pk primary key(a);
Query OK, 0 rows affected (0.57 sec)
```

7. Create table t2 with referential integrity:
field b is foreign key referencing field a of table t1;

```
mysql> create table t2 ( b int references t1(a), c text , d date ) ;
Query OK, 0 rows affected (0.17 sec)
```

OR

```
mysql> create table t2 (b int , c text , d date, foreign key(b) references t1(a)) ;
```

8. If foreign key is not specifed at the time of table creation , add foreign key contraint by giving :

```
mysql> alter table t2 add constraint fk1 foreign key(b) references t1(a);
```

fk1 is the constraint name.

9. To create foreign key contraint with on delete and on update cascade

```
mysql>create table t2 ( b int , c text , d date, foreign key(b) references t1(a) on delete
cascade on update cascade) ;
```

10. To see the tables created under the database

```
mysql> show tables;
```

11. To see the description of a table :

```
mysql> describe <table_name>;
```

12. To the create table description of a table :

```
mysql>show create table <table_name> \G;
```

13. To add unique constraint :

```
mysql> create table t1( a int  unique , b text);
```

14. Field with default value:

```
mysql> create table t1( a int  default 0  , b text);
```

15. To drop a database:

```
mysql > drop database <database_name>;
```

16. To drop a table:

```
mysql > drop table <table_name>;
```

17. To see mysql help

```
mysql> help contents
                                OR
mysql> \h
                                OR
mysql> help  <command_name>
```

MySQL Datatypes

TEXT TYPES

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.
LONGBLOB	A string with a maximum length of 4294967295 characters.

The ( ) brackets allow you to enter a maximum number of characters will be used in the column.
VARCHAR(20)

CHAR and VARCHAR are the most widely used types. CHAR is a fixed length string and is mainly used when the data is not going to vary much in it's length. VARCHAR is a variable length string and is mainly used when the data may vary in length.

CHAR may be faster for the database to process considering the fields stay the same length down the column. VARCHAR may be a bit slower as it calculates each field down the column, but it saves on memory space. Which one to ultimatly use is up to you.

Using both a CHAR and VARCHAR option in the same table, MySQL will automatically change the CHAR into VARCHAR for compatability reasons.

BLOB stands for Binary Large OBject. Both TEXT and BLOB are variable length types that store large amounts of data. They are similar to a larger version of VARCHAR. These types can store a large piece of data information, but they are also processed much slower.

NUMBER 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 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.

The integer types have an extra option called unsigned. Normally, the integer goes from an negative to positive value. Using an UNSIGNED command will move that range up so it starts at zero instead of a negative number.

DATE TYPES

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

MISC TYPES

ENUM ( )	Short for ENUMERATION which means that each column may have one of a specified possible values.
SET	Similar to ENUM except each column may have more than one of the specified possible values.

ENUM is short for ENUMERATED list. This column can only store one of the values that are declared in the specified list contained in the ( ) brackets.
ENUM('y','n')
You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted.
SET is similar to ENUM except SET may contain up to 64 list items and can store more than one choice.

Use of auto_increment data attribute
Name: 'AUTO_INCREMENT'
Description:
The AUTO_INCREMENT attribute can be used to generate a unique identity for new rows:

Example:
```
CREATE TABLE animals (
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (id)
 );


INSERT INTO animals (name) VALUES
   ('dog'),('cat'),('penguin'),
   ('lax'),('whale'),('ostrich');
SELECT * FROM animals;
```
NOTE:
For all the databases described below, apply
- Primary Key constraint
- Foreign key constraint along with on delete cascade and on update cascaded.
- Check the delete cascading, update cascading for delete and update queries respectively.
- Use appropriate datatypes.

DATABASES:

1) Bank Database

Consider the following database of Bank. A bank maintains the customer details, account details and loan details. It has the Branch information also.

Following are the tables:

ACCOUNT(<u>ACC_NO</u> INT, ACC_TYPE CHAR(10), BALANCE FLOAT(8,2))

LOAN(<u>LOAN_NO</u>INT, LOAN_AMT DOUBLE(9,2) , NO_OF_YEARS INT)

BRANCH(<u>BRANCH_NO</u> INT, BRANCH_NAME CHAR(20), BRANCH_CITY VARCHAR(20))

CUSTOMER(<u>CUST_NO</u> INT , CUST_NAME CHAR(20), CUST_STREET CHAR(15), CUST_CITY VARCHAR(20))

The relationships are as follows. :-
CUSTOMER-ACCOUNT: 1-M
CUSTOMER- LOAN: 1-M
BRANCH-LOAN: 1-M
BRANCH-ACCOUNT: 1:M

Constraints:
1) use auto_increment data type attribute for cust_no
2) branch_name should be not null.

2) Bus transport System

Consider the following database of Bus transport system . Many buses run on one route. Drivers are allotted to the buses shiftwise.

Following are the tables:

BUS (<u>BUS NO</u> INT , CAPACITY INT , DEPOT_NAME VARCHAR(20))
ROUTE (<u>ROUTE NO</u> INT, SOURCE CHAR(20), DESTINATION CHAR(20), NO_OF_STATIONS INT)
DRIVER (DRIVER_NO INT , DRIVER_NAME CHAR(20), LICENSE_NO INT, ADDRESS CHAR(20), D_AGE INT , SALARY FLOAT)

The relationships are as follows:

BUS_ROUTE : M-1
BUS_DRIVER : M-M with descriptive attributes Date of duty allotted and Shift – it can be 1 (Morning) 0r 2 (Evening).

Constraints :
1. License_no is unique.
2. Bus capacity is not null.

3) Student- Competition database

Consider the following database maintained by a school to record the details of all the competitions organized by the school every year. The school maintains information about students as well as competitions. Competition type can be like ‘academics’ or ‘sports’ etc.

Following are the tables:

STUDENT (<u>SREG NO</u> INT , NAME CHAR(30), CLASS CHAR(10))
COMPETITION(<u>C NO</u> INT , NAME CHAR(20), TYPE CHAR(15))

The relationship is as follows:
STUDENT-COMPETITION: M-M with described attributes rank and year.

4) Client-Policy Database

Consider an insurance company which has agents. Clients select a particular policy ang go for the policy through the agents. Company manintains information about the clients and agents . Whenever client takes a policy , agent validates the information of client such as age of the client should be in the range of the selected policy(i.e. Age should be between minimum_age_limit and maximum_age_limit.) , sum_assured also should be between the min_sum_assured and max_sum_assured. The client gets a unique policy number , decides the premium amount , type_of_premium , nominee name etc. The policy term is calculated as the maturity age of the selected policy – age of the client.

POLICY(<u>POLICY_NAME</u>VARCHAR(20) , MIN_AGE_LIMIT INTEGER , MAX_AGE_LIMIT INTEGER , MATURITY_AGE INTEGER , MIN_SUM_ASSURED INTEGER , MAX_SUM_ASSURED INTEGER) ;

CLIENT(<u>CLIENT_ID</u>INTEGER , NAME VARCHAR(25) , BIRTH_DATE DATE ,NOMINEE_NAME VARCHAR(25) ,RELATION_WITH_CLIENT VARCHAR(20));

AGENT(<u>AGENT_ID</u>INTEGER , NAME VARCHAR(25), LICENSE_NO INTEGER , BRANCH_OFFICE VARCHAR(20)) ;

Relationship between :

POLICY , CLIENT and AGENT is ternary with described attributes POLICY_NO, PREMIUM AMOUNT , POLICY_DATE , TYPE_OF_PREMIUM , SUM_ASSURED and POLICY TERM.

The relationship table is:

AGENT_CLIENT_POLICY(AGENT_ID INTEGER , CLIENT_ID INTEGER , POLICY_NAME VARCHAR (20) , POLICY_NO INTEGER , PREMIUM DECIMAL(7,2) , POLICY_DATE DATE , TYPE VARCHAR(20) , SUM_ASSURED DECIMAL(7,2) , TERM INTEGER).

type : is the type of premium which can be 'q' (quarterly),'h' (half yearly) ,'y' (Yearly).While inserting records in relationship table enter the type value as one of the 'q', 'h','y'.

Constraints:

1.use auto_increment data type attribute for client_id.
2. policy_no is unique.

5) Real Estate Database

Consider the Real Estate Agency Database where estate agents are sold by many agents. Estates are purchased by Customers from Agents and agent will get a commission.

Following are the tables:

1. AGENT(<u>AID</u> INT , ANAME VARCHAR(20) ,ADDRESS VARCHAR(20), CONTACTNO VARCHAR(10));
2. ESTATE(<u>ENO</u> INT, TYPE VARCHAR(20),LOCATION VARCHAR(20), PRICE INT) ;
3.CUSTOMER(<u>CUSTID</u> INT, CNAME VARCHAR(20), CONTACTNO VARCHAR(10),ADDRESS VARCHAR(30));

Type : Estate type can be 1bhk flat , land, 2bhk flat etc.

The Relationship Between :

Agent, Estate and Customer is a ternary relationship. A relationship table Transaction will store the transaction about customer purchased estate from agent. A customer can purchase many estates from one or more agents .

A transaction table is:

TRANSACTION(AID ,ENO, CUSTID, PURCHASEDATE DATE, COMMISSION DECIMAL(5,2))

Constraints :

1. Not null on price
2. Unique constraint on Eno on transaction table (one estate sold to only one customer)

6) Mobile Billing Database

Consider a database of Gigabyte Mobile Services which provide postpaid services to the customers. Different service plans are available from which a customer can select any one. The monthly customer call information is recorded (in custcallinfo table) and a bill is generated at the end.

Following are the tables:

1. PLAN(<u>PLANNO I</u>NT, PNAME VARCHAR(20), NOOFFREECALLS INT, RATEPERCALLPERMIN DOUBLE, FREECALLTIME INT, FIXEDAMT DOUBLE);

2.CUSTOMER<u>(CUSTNO</u> INT,NAME VARCHAR(20),MOBILENO VARCHAR(10));

3. CUSTCALLINFO(<u>RECNO</u> <u>I</u>NT,CUSTNO INT,NOOFCALLS INT , TOTALTAKLTIME INT , CYCLEPERIOD VARCHAR(20))

4. BILL(RECNO INT,<u>BILLNO </u>INT,FINALBILLAMT DOUBLE,CYCLEDATE DATE , BILLDUEDATE DATE,STATUS VARCHAR(10),BILLPAYDATE DATE)

CYCLEPERIOD: shows the period between two particular months . e.g 'jan-feb' , 'oct-nov'.
CYCLEDATE: is in between the respective cycle period.

Following are the relationships:

1.PLAN-CUSTOMER : 1-M
2.CUSTOMER-CUSTCALLINFO : 1-M
3.CUSTOMER-BILL :1-1

Constraint :

1. FIXEDAMT in plan is by default 0.

7. Railway Reservation System

Consider a railway reservation system of passengers. Passengers reserve berths of a bogie of trains. The bogie capacity of all the bogies of a train is same.

1. TRAIN (TRAIN_NO INT, TRAIN_NAME VARCHAR(20), DEPART_TIME TIME , ARRIVAL_TIME TIME, SOURCE_STN VARCHAR(20) , DEST_STN VARCHAR (20), NO_OF_RES_BOGIES INT , BOGIE_CAPACITY INT)

2. PASSENGER (PASSENGER_ID INT, PASSENGER_NAME VARCHAR(20), ADDRESS VARCHAR(30), AGE INT , GENDER CHAR)

Relationship is as follows:

TRAIN _PASSENGER : M-M with descriptive attributes as follows :
TICKET (TRAIN_NO INT , PASSENGER_ID INT, TICKET_NO INT COMPOSITE KEY, BOGIE_NO INT, NO_OF_BERTHS INT , DATE DATE , TICKET_AMT DECIMAL(7,2),STATUS CHAR)

The status of a particular berth can be ‘W‘ (waiting) or ‘C‘ (confirmed).

8). Sales Management System.

Consider a shop that stores the data about the products ,employees and customers . The employees show the products to the customers and service them . A customer may buy many products from many employees . Whenever a customer buys a product , (or products) the transaction is recorded along with the delivery-date , transaction-date and the total amount. The transactionID is unique.

<u>Createthedatabasefortheabovesystem .</u>

CUSTOMER- MASTER : - The table is a master table which will hold the information of customers who have bought a product from the store .

CUSTOMER(<u>CUST ID</u> VARCHAR(6) , CUST_NAME VARCHAR(30) , ADDRESS VARCHAR(40) , CONTACT_NO INTEGER);

EMPLOYEE - MASTER : - The table is a master table which will hold the details of the employees working in the store .

EMPLOYEE(<u>EMP ID</u> VARCHAR(6) , EMP_NAME VARCHAR(25) , DESIGNATION VARCHAR(20), MANAGER_ID INT);

PRODUCT - MASTER: - The table is a master table which will hold the details of the products available in the store for sale.

PRODUCT_MASTER(PRODUCT_ID VARCHAR(6) , PROD_NAME VARCHAR(25) , COMPANY_NAME VARCHAR(20) , UNIT_COST_PRICE DECIMAL (7,2) , QTY_IN_HAND INTEGER) ;
Company name is the name of company manufacturing or trading this product
The relationship between CUSTOMER and EMPLOYEE: M -M.

Consider the relationship table TRANSACTION which will hold the details of the transactions of the sale with described attributes TRANSACTIONID , TRANSACTION-DATE, DELIVERY-DATE , AMOUNT.

TRANSACTIONID	Identity no of the transaction (primary key)
TRANSACION-DATE	Date on which the transaction is performed.
DELIVERY-DATE	Date on which the product is delivered.

AMOUNT	Total amount in terms of all the products purchased .

TRANSACTION AND PRODUCT:M-M.
Consider the relationship table TRANSACTION_PRODUCT which will hold the details of the transactions with the described attribute quantity ordered.
Constraints:
PROD_NAME - NOT NULL
UNIT_COST_PRICE & QTY_IN_HAND are by default 1.
MANAGERID may be NULL (Identity number of the manager to whom the employee reports to)

End of Session

Session 1: Queries

Start Date __/_____/_____

Objectives

To learn about:

- DML statements and functions.

Reading

The students should be aware of creating databases, tables and inserting records in the tables.

Ready References

1. Retrieving Information from a Table

SELECT: The SELECT statement is used to pull information from a table.

The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

what_to_select indicates what you want to see. This can be a list of columns, or * to indicate ‘‘all columns.“

which_table indicates the table from which you want to retrieve data.

WHERE clause is optional, if it is present, conditions_to_satisfy specifies conditions that rows must satisfy to qualify for retrieval.

Limit Clause :

The LIMIT ‘n‘ returns the first ‘n‘ rows of a result set.

```
mysql> select cid,cname, caddress from customer limit 2 ;
+-----+--------+-----------+
| cid | cname  | caddress  |
+-----+--------+-----------+
| 11  | shilpa | warje     |
| 12  | radha  | nana peth |
+-----+--------+-----------+
2 rows in set (0.01 sec)
mysql> select cid,cname, caddress from customer limit 2, 5;
+-----+-------+------------+
| cid | cname | caddress   |
+-----+-------+------------+
| 13  | amol  | kothrud    |
| 14  | amit  | dhankawadi |
| 15  | sumit | dhankawadi |
+-----+-------+------------+
3 rows in set (0.00 sec)
```

2. DELETE: Delete is used to delete records from the table.

```
mysql> DELETE FROM table_name where condition;
```

3. UPDATE: Update is used to modify a particular record.

```
Mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
```

4.Sorting Rows

To sort a result, use an ORDER BY clause.
Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```

Th default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the DESC keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
```

5. Date and time functions

** CURDATE()

Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
    -> '2008-06-13'
mysql> SELECT CURDATE() + 0;
    -> 20080613
```

CURRENT_DATE and CURRENT_DATE() are synonyms for CURDATE().

** CURTIME()

Returns the current time as a value in 'HH:MM:SS' or HHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

```
mysql> SELECT CURTIME();
    -> '23:50:26'
mysql> SELECT CURTIME() + 0;
    -> 235026.000000
```

CURRENT_TIME and CURRENT_TIME() are synonyms for CURTIME()

** DATE(expr)

Extracts the date part of the date or datetime expression expr.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
        -> '2003-12-31'
```

** DATEDIFF(exprl,expr2)

DATEDIFF() returns expri – expr2 expressed as a value in days from one date to the other. expri and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
    -> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
    -> -31
```

** DAYNAME(date)

Returns the name of the weekday for date.

```
mysqi> SELECT DAYNAME('2007-02-03');
        -> 'Saturday'
```

** DAYOFMONTH(date)

Returns the day of the month for date, in the range 1 to 31, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero day part.

```
mysqi> SELECT DAYOFMONTH('2007-02-03');
        -> 3
```

DAYOFWEEK(date)

Returns the weekday index for date (1 = Sunday, 2 = Monday, …, 7 = Saturday). These index vaiues correspond to the ODBC standard.

```
mysqi> SELECT DAYOFWEEK('2007-02-03');
        -> 7
```

** DAYOFYEAR(date)

Returns the day of the year for date, in the range 1 to 366.

```
mysqi> SELECT DAYOFYEAR('2007-02-03');
        -> 34
```

** EXTRACT(unit FROM date)

The EXTRACT() function uses the same kinds of unit specifiers as DATE_ADD() or DATE_SUB(), but extracts parts from the date rather than performing date arithmetic.

```
mysqi> SELECT EXTRACT(YEAR FROM '2009-07-02');
       -> 2009
mysqi> SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
       -> 200907
mysqi> SELECT EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03');
       -> 20102
mysqi> SELECT EXTRACT(MICROSECOND
    ->              FROM '2003-01-02 10:30:00.000123');
        -> 123
```

** MONTH(date)

Returns the month for date, in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part.

```
mysqi> SELECT MONTH('2008-02-03');
        -> 2
```

** MONTHNAME(date)

Returns the fuii name of the month for date.

```
mysqi> SELECT MONTHNAME('2008-02-03');
        -> 'February'
```

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, compute the difference in the year part of the current date and the birth date, then subtract one if the current date occurs earlier in the calendar year than the birth date. The following query shows, for each pet, the birth date, the current date, and the age in years.

```
mysql> SELECT name, birth, CURDATE(),
-> (YEAR(CURDATE())-YEAR(birth))
-> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
-> AS age
-> FROM pet;
```

Here, YEAR() pulls out the year part of a date and RIGHT() pulls off the rightmost five characters that represent the MM-DD (calendar year) part of the date. The part of the expression that compares the MM-DD values evaluates to 1 or 0, which adjusts the year difference down a year if CURDATE() occurs earlier in the year than birth. The full expression is somewhat ungainly, so an alias (age) is used to make the output column label more meaningful.

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
```

Finding animals with birthdays in the upcoming month is easy, too. Suppose that the current month is April. Then the month value is 4 and you look for animals born in May (month 5) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

6. String Functions

TRIM : Removes all specified characters either from beginning or end of the string

```
mysql> select trim(‘     bar     ‘);
    ‘bar’
mysql>select trim(leading ‘x’ from ‘xxxbarxxx’);
      ’barxxx’
mysql>select trim(trailing ‘xyz’ from ‘barxxyz’);
    ‘barx’
mysql>select trim(both ‘x’ from ‘xxxbarxxx’);
    ‘bar’
```

Rest of the function description you can get by giving the help command at mysql prompt

```
mysql> help <function_name>
```

e.g. Myql> help substring

Practical Assignment

SET A

1. Using Bank Database

1. Find out customer name having loan amt >10000
2. Select customers having account but not loan.
3. Select customers having account as well as loan.
4. Find out customer names having loan at ‘Pimpri’ branch.
5. Find out customer names having Saving account.
6. Find out the total loan amount at Nagar Branch.
7. List the names of customers who have taken loan from the branch in the same city they live.

2. Using Bus Database

1. Find out the drivers working in shift 1.
2. Find out the route details on which buses of capacity 20 runs.
3. Find the names and their license no. of drivers working on 12-01-2008 in both the shifts.
4. Delete all the routes on which total stations are less than 3.
6. Find out the number of buses running from ‘Chichwad‘ to ‘Corporation’.
7. Update the salary of driver by 1000 if his age > 35.
8. List the bus numbers which are running from ‘Swargate’ to ‘Hadapsar’ having bus capacity 50.

3. Using Student Competition Database

1. List out all the competitions held in the school.
2. List the names of all the students who have secured 1st rank in running race from 1995 to 2005.
3. Give the name of a student who has won maximum number of competitions.
4. Find out the total number of competitions organized in the school for competition type ‘sports‘.
5. Find out the details of students participating in different competitions.

SET B

1 . Using Client Policy Database

1. Count the number of clients who have taken policies from branch office ‘Pune‘
2. Give the name of agent having maximum count of clients.
3. Find the name of clients who have taken j-b policy and premeum if hals yearly on 1st march, (year does not matter .).
4. Count the number of clients of ‘j-a’ policies from ‘mumbai-1’ branch office.
5. Find the total premium amount of client ‘_______’.

2. Using Real Estate Database
1. Find the name of agents with their contact numbers who have sold 2BHK flats in F.C.

road area location.
2. Give the agent wise list of customers.
3. Find the total commission earned by Amit by selling Land.
4. Give the names of customer who have purchased an estate in the same area in which they are living.

3. Using Mobile Billing Database

1. Find the total bill collection of all the customers for the cycle Nov.-Dec.;
2. Find the details of bills which are paid before billduedate;
3. Find the names of the customers who have paid the bill within due date.
4. Write a query to delete all customer call info if the customer is deleted.
5.Use the following mysql functions :
   a. Round of finalbillamt
   b. Extract year and month from cycle date
   c. Upper and Lower case on customer name;
   d. Trim specific characters from beginning and end
   e. Concat two strings together.

SET C

1. Using Railway Reservation Database

1) List the information about the availability of trains between "Mumbai" and "Pune".
2) Give the names of all the trains which start from 'Pune'.
3) Give the details of the passengers who have done reservation for the train named "Shatabdi Express".
4) Display ticket wise number of berths reserved on date = '-------'.
5) Display train wise ticket amount on date = '-------'.
6) Delete the details of all the passengers who have booked the tickets before today's date.

2. Using Sales Management Database

1. Give the names of customers who have bought the products whose name contains exactly three characters along with product cost. (use regexp).
2. List the name of company whose products are not transacted on date = '___________'.
3 Update the amount field of transaction table to total amount of all the products purchased.
4 List the names of products of company 'appollo' Mr. Patil has bought .
5. Show the details of first three employees of emp_master table.
6. Show the first two product names that 'Mr.Patil' has bought in descending order of product cost.
7. List the names of customers who have bought products having substring 'plus' in product names.

Signature of the Instructor   Date of Completion ____/____/_____

End of Session

Session 2: Stored Functions

Start Date __/_____/_____

Objectives

To learn about:

- Demonstrate the use of functions.

Reading

You should read the following topics before starting the exercise

- Declaring and Defining a function
- How to call a function
- How to pass parameter to the function
- Function returning a value

Ready References

You should already be comfortable with the concept of subroutines and functions : encapsulated pieces of code that can be called by programs often used to carry out repetitive or complicated task. You have already used standard functions in sql . User can write their own stored function which is stored program that returns a value.

STORED FUNCTIONS

1. You can create a stored function using the following syntax:

```
CREATE FUNCTION function_name (function_parameter [,…….] )
RETURNS datatype
[LANGUAGE SQL]
[[NOT] DETERMINISTIC]
[{CONTAINS SQL | NO SQL | MODIFIES SQL DATA | READS SQL DATA)]
[SQL SECURITY { DEFINER | INVOKER)]
[COMMENT comment_string]
function_statements
```

The RETURNS is mandatory and defines the data type that the function will return.

LANGUAGE SQL :

At present, SQL is the only supported store procedure language, but there are plans to introduce other languages in future like Perl , Python etc.

[NOT] DETERMINISTIC

Only used with stored functions , any function is declared as DETRMINISTIC will return the same value every time, provided the same set of parameters is passed in. Declaring a function DETRMINISTIC helps MySQL optimize execution of the stored function.

CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA

This setting indicates what type of task the stored procedure will do. The default, CONTAINS SQL, specifies that SQL is present but it will not read or write data. NO SQL indicates that no SQL is present in the procedure. READS SQL DATA indicates that the SQL will only retrieve data. Finally, MODIFIES SQL DATA indicates that the SQL will modify data. At the time of writing, this characteristic had no bearing on what the stored procedure was capable of doing.

SQL SECURITY {DEFINER I INVOKER)
If the SQL SECUTRITY characteristic is set to DEFINER, then the procedure will be executed in accordance with the privilege of the user who defined the procedure. If set to INVOKER, it will execute according to the privileges of the user executing the procedure.

COMMENT string
You can add some descriptive information about the procedure/function by using COMMENT characteristic.

Function Statements: Valid SQL function statement.

2. Executing a Stored Function

The select statement invokes a function that was defined previously with CREATE function.. Syntax : select function_name([parameter[, ...]]);

You can use sql language braching in functions:

IF expri
THEN
...
ELSEIF
expr2
THEN
...
ELSE
...
END IF
	CASE expr
	WHEN vali
	THEN ...
	WHEN val2
	THEN ...
	ELSE ...
	END CASE
		foo:
		LOOP
		... LEAVE
		foo; ...
		END LOOP
		foo
		
		(foo is label
		name)
			WHILE expression DO
			...
			END WHILE
			
			foo
			REPEAT
			... ITERATE foo; ...
			UNTIL expression END
			REPEAT

Sample Code :

The following are the examples of simple functions that takes a parameter ,perform an operation using SQL functions and returns the result. Before writing a function you have to use a database as :
e.g mysqi > use database <u>first</u>

where <u>first</u> is the database which is aiready created by you on which you want to write a function.
One other thing to bear in mind is that semicolons (;) are used as a part of the definition of the function. For this reason the end of line must be redefined to something that won't be used in the definition.
For example:

```
delimiter //
```

1. A sample function for how to calculate a circumference of circle .

```
   mysql> delimiter / /
   mysql> create function circum1(r double) returns double
   -> deterministic
   -> begin
   -> declare c double;
   -> set c = 2 * r * pi();
   -> return c;
   -> end
   -> / /

Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> select circum1(4);
+-----------------------+
| circum1(4)           |
+-----------------------+
| 25.132741228718 |
+------------------------+
1 row in set (0.00 sec)
```

2. A Sample function using RETURN statement which accepts an input parameter marks and returns the result .

```
  delimiter / /
create function exam_stat1(marks int)
returns varchar(20)
deterministic
begin
declare result varchar(20);
     if marks >70 and marks < 100 then
          set result = 'distinction';
    elseif marks >60 and marks <70 then
          set result = 'first class';
    elseif marks >50 and marks <60 then
          set result = 'second class';
    elseif marks > 40 and marks <50 then
          set result = 'pass class';
    end if;
    return(result);
    end
 / /



mysql> \! vi exam_stat1
mysql> \. exam_stat1
Query OK, 0 rows affected (0.00 sec)
```

```
 mysql> select exam_stati(80);
       -> / /
         +----------------------+
          I exam_stati(80) I
         +----------------------+
          I      distinction    I
          +---------------------+

     i row in set (0.00 sec)
mysql> delimiter ;
 mysql> select exam_stati(55);
         >/ /
           +---------------------+
            I exam_stati(55) I
           +---------------------+
            I second class     I
           +----------------------+
          i row in set (0.00 sec)

 mysql> select exam_stati(45);
             +----------------------+
              I exam_stati(45) I
             +--------------------- +
              I pass class         I
             +-----------------------+
            i row in set (0.00 sec)
```

3. A sample code of using SQL statements in stored function

```
     delimiter / /
     create function samplefucn() returns int
     deterministic
     begin
     declare rows int;
     select count(*) into rows from bill;
     return rows;
     end
     / /
     mysql> \! vi samplefucn
     mysql> \. samplefucn
            Query OK, 0 rows affected (0.00 sec)
     mysql> select samplefucn();
            -> / /
               +------------------+
               I samplefucn() I
               +------------------+
               I          3        I
               +------------------+

     i row in set (0.i2 sec)
```

Self Activity

1. Write functions for the following
   i) To find the area of circle
   ii) To find the area of rectangle.
2. Write the function using various simple SQL statements.
3. Write the functions which uses input parameter / s.

Practical Assignment

SET A

1. Using Bank Database
a) Write a function that returns the total loan amount of a particular branch.(Accept branch name as input parameter.)

b) Write a function to count the no. of customers of particular branch.

(Accept branch name as input parameter).

2. Using Bus Transportation Database

a) Write a function to find out the name of the driver having maximum salary.
b) Write a function to accept the bus_no and date and print its allotted driver.

3. Using Student Competition Database
a) Write a function which accepts a competition type and returns the total no of competitions held under that type.
b) Write a function which accepts a name of students and returns the total no of prizes won by that student in the year 2001.

SET B

1. Using Client Policy database
a) Write a function to accept an input parameter p and return the following
   'yearly'   if p is 'y' ,'
   'half yearly' if p is 'h' and
   'quarterly'   if p is 'q'.
The return datatype and datatype of p are similar to the datatype of type_of_premium field of relation table.(Use substring function)

b) Write a function to print the name of the client whose sum assured is maximum among all the clients of policy = '_____'. (Assume only one client with maximum sum assured.)

2. Using Real Estate Agency Database
a) Write a function to calculate the total price of estates the particular agent had sold.
   (accept agent name as an input parameter)
b) Write a function which will find the location from which max estates are sold.

3. Mobile Billing Database

a) Write a function which will find the total no.of calls of all the customers in the cycle period ‘jan-feb‘
b) Write a function to count the no of customers for the given plan .(Accept plan name as a input parameter)

SET C

1. Using Railway Reservation Database

a) Write a function which accepts train name and date as a input parameter and calculate total ticket amount for that train on the given date.
b) Write a function to calculate total no of berths reserved for the train ‘Shatabdi Express‘ on 2009-05-18.

2. Using Sales Management Database

a) Write a function to count the number of customers who have bought fridge of company xyz on 1st Jan 2000 . (pass parameters product_name, company name and date to function and return count.)
b) Write a function to give the total sale of the employee in the month of march in year 2009. (pass emp_id , month and year parameters to the function and return the total sale.).

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2.Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of the Instructor Date of Completion ____/____/_____

End of Session

Session 3: Stored Procedures

Start Date __/_____/_____

Objectives

To learn about:

- Usage of stored procedures.

Reading

You should know the stored functions before starting the exercise.

Ready References

Stored procedures are collections of SQL commands and program logic stored on the database server. A stored procedure is a set of SQL statements that are stored in the server and executed as a block.
Stored procedures allow most database access logic to be separated from the application logic. One of the indirect benefits of using stored procedures is that application code becomes smaller and easier to understand. Another advantage of stored procedures is that the SQL can be "pre-compiled" increasing the speed of the application.

Create procedure syntax.

```
CREATE
[DEFINER = { user I CURRENT_USER }] PROCEDURE sp_name
([proc_parameter[,...]])
[characteristic ...] routine_body
proc_parameter:
[ IN I OUT I INOUT ] param_name type
type:
Any valid MySQL data type
characteristic:
LANGUAGE SQL I [NOT] DETERMINISTIC I I CONTAINS SQL I NO SQL I
READS SQL DATA I
MODIFIES SQL DATA I I SQL SECURITY I DEFINER I INVOKER I I COMMENT
'string'
routine_body:
Valid SQL procedure statements
```

Executing a Stored Procedure
The CALL statement invokes a procedure that was defined previously with CREATE PROCEDURE.. Syntax : CALL sp_name([parameter[, ...]]);

Syntax of different statements

CREATE PROCEDURE	Creates a stored procedure, which is stored in the proc table in the MySQL database.
ALTER PROCEDURE	Alters a previously defined stored procedure that was created using the CREATE PROCEDURE statement. It does not affect related stored procedures or stored functions.
DROP PROCEDURE	Removes one or more stored procedures from MySQL's proc table.
SHOW CREATE PROCEDURE	Returns the text of a previously defined stored procedure that was created using the CREATE PROCEDURE statement. This statement is a MySQL extension to the SQL:2003 specification.
SHOW PROCEDURE STATUS	Returns the characteristics of a previously defined stored procedure; including name, type, creator, creation date, and modification date. This statement is a MySQL extension to the SQL:2003 specification.
CALL	Invokes a previously defined stored procedure that was created using the CREATE PROCEDURE statement.
BEGIN ... END	Contains a group of multiple statements for execution.
DECLARE	Used to define local variables, conditions, handlers, and cursors.
SET	Used to alter the values of both local variables and global server variables.
SELECT ... INTO	Used to store the indicated columns directly into variables.
IF	An if-then-else-end if condition statement.
CASE ... WHEN	A case statement conditional construct.
LOOP	A simple looping structure; exiting is performed using the LEAVE statement.
LEAVE	Used to exit IF, CASE, LOOP, REPEAT and WHILE statements.
ITERATE	Used within loops to restart at the beginning of the loop.
REPEAT	A loop with the conditional test at the end.
WHILE	A loop with the conditional test at the beginning.

Creating dynamic procedures
The real benefit of stored procedure is when values are passed and received back.
There are three types of parameters .
IN : The default. This parameter is passed to the procedure and can be changed inside the procedure , but remains unchanged outside .
OUT :- No value is supplied to the procedure (it is assumed to be null) but it can be modified inside the procedure and is available outside the procedure.
INOUT :- The characteristics of both IN and OUT parameters. A value can be passed to the procedure , modified there as well as passed back again.
IN:- Is default parameter for stored procedure in case the parameter type is not specified.

Solved Examples:-

Consider customer table with fields
Customer (cust_id int , cust_name varchar(2O) , address text);

1. Create a procedure to delete the customer records based on cust_id

```
Delimiter / /
Create procedure delete_records(custID int )
  Begin
        Delete from customer  where cust_id = custID;
  End
/ /
Delimiter ;
```

Execute the procedure as :- call delete_records(1O) and verify the deletion.

2. Create a procedure to return the total number of customers .

```
Delimiter / /
Create procedure display(out para1 int)
Begin
       Select count(*)  into para1 from customer ;
End
/ /
Delimiter ;
```

Execute the procedure as :- call display (@No_of_cust) .
Display the value held in the variable No_of_cust as
Select @No_of_cust;

Create a procedure named square which will accept an integer value as a parameter and return the square root of the same.

Create procedure square(inout p int) set p = p* p;

Execute the procedure as -
Set @number=1O;
Call square(@number);
Display the value of p as :- select @number ;

Self Activity

1. Create a stored procedure named display message which will display the message ‘Welcome ro RDBMS world’.
2. Create a stored procedure named addrecords for adding employee records in the table.
3. Create a procedure named employeedata which will retrive the name and address of employee from the employee table for a given employeeID passed as a parameter.
4. Drop the procedure called displaymessage.

Practical Assignment

SET A

1. Using Bank Database

a) Write a procedure to transfer amount of 1000 Rs. from account_no 10 to account_no 20.
b) Write a procedure withdrawal for the following
1. Accept balance amount and acc_no for withdrawal as input parameters.
2. Check if input amount is less than actual balance of accounts.
3. If input amount is less, give the message “withdrawal allowed from account” otherwise give the message “withdrawal not allowed from account”. Update the balance field.

2. Using Bus transportation Database

a) Write a procedure to list the drivers working on a particular date shift wise.Accept the date as an input parameter.
b) List the bus_no and drivers allocated to the buses which are running from ‘Kotharud’ to ‘Nigdi’ on date ________.

3. Using Student Competition Database

a) Write a procedure to count the no of competitions which come under the type ‘sports‘ and no of competitions which come under the type ‘academics‘.
b) Write a stored procedure which accepts year as input and gives a list of all competitions held in that year.

SET B

1. Using Client Policy Database

a) Create a procedure to update the premium of a policy_number of a client . If the premium amount exceeds the maximum sum assured of the policy , rollback the transaction else commit.
b) Write a procedure to print the names of agents who have more than or equal to 4 policies also who have less than 4 policies in the year y.
(The input parameter y is the year .)

2. Using Real Estate Agency Database

a) Write a procedure which will list all estate type with their agent name on particular location (Accept location as an input parameter)
b) Write a procedure which will find the estates having maximum price among each agent type.

3. Using Mobile Billing Database

a) Write a procedure which will accept plan name as an input and list names of customers who took that plan.
b) Write a procedure which will give the names of the customers along with their final bill amount and bill due date.

SET C

1. Railway Reservation Database

a) Write a procedure to calculate the ticket amount paid by all the passengers on 13/5/2009 for all the trains.
b) Write a procedure to update the status of the ticket from "waiting" to "confirm" for passenger named "Mr. Jadhav"

2. Sales Management System
a) Create a procedure to print the names of customers who have bought at least one product on 1st December 2005. (i.e. delivery date is 1st December 2005.)
b) Create a procedure to print the total sale in January month 2006 under the manager 'Mr. Joshi.'(Use extract function to handle month and year).

Assignment Evaluation

0: Not Done [ ]	1:Incomplete [ ]	2.Late Complete [ ]
3:Needs Improvement [ ]	4:Complete [ ]	5:Well Done [ ]

Signature of the Instructor   Date of Completion ____/____/______

End of Session

Session 4: Cursors

Start Date __/_____/_____

Objectives

To learn about:

- Use of cursors.

Reading

You should be familiar with the concept of procedures.

Ready References

Mysql engine uses a work area for its internal processing in order to execute an SQL statement . This work area is private to SQL's operations and is called as cursor.The values retrieved from a table are held in the cursor opened in memory by the MySQL Db engine.

Cursors are named result sets , which can be processed within procedures.

Cursors are supported inside stored procedures , functions and triggers. Cursors in MySQL have these properties: Cursors must be declared before declaring handlers. Variables and conditions must be declared before declaring either cursors or handlers.

When indivisual records in a table have to be processed inside a code block , a cursor is used. This cursor will be declared and mapped to an SQL query in the Declare Section of the code block and used within its Executable Section, A cursor thus created and used is known as an Explicit Cursor.

<u>Cursor Declaration .</u>A cursor is defined in the declarative part of a code block.This is done by naming the cursor and mapping it to the query.

Syntax.

```
DECLARE <cursor name > CURSOR FOR <Select statement>
```

Declarations must follow a certain order. Cursors must be declared before declaring handlers. The handler section of a cursor handles the errors or execptions using following syntax.

```
DECLARE <handler type > HANDLER FOR <conditionvalue>[,...] <statement>
```

Cursor OPEN Statement

```
OPEN cursor_name
```

This statement opens a previously declared cursor.

Cursor FETCH Statement

FETCH cursor_name INTO var_name [, var_name] ...

This statement fetches the next row (if a row exists) using the specified open cursor, and advances the cursor pointer.

If no more rows are available, a No Data condition occurs with SQLSTATE value 02000. To detect this condition, you can set up a handler for it (or for a NOT FOUND condition).

Cursor CLOSE Statement

CLOSE cursor_name

This statement closes a previously opened cursor.

If not closed explicitly, a cursor is closed at the end of the compound statement in which it was declared.

Example: Refer this code to understand the use of syntax.

Consider a table test(a int , b float(4,2) , c text);

Create a procedure using cursor to display the records of table test .

```
create procedure print_recirds ()

begin

declare done int default  0;

 declare al int ;

 declare bl float ;

 declare cl text;

 declare curl cursor for select a , b, c from test ;

declare continue handler for SQLSTATE ‘02000‘ set done = l;

open curl ;

repeat

 fetch curl into al,bl,cl;

 if not done then

select al , bl , cl;

end if;
```

```
until done end repeat ;

close cur1; end / /
```

Self Activity

1. Execute the above cursor .

2. In the above example use two different cursors . Cur1 will store values of a and b b in table t1 . Cur2 will store the value of c in table t2.

3. Try it on different tables using different queries.

Practical Assignment

SET A

1. Using bank database

a) Write a procedure using cursor to display the customers having loan amounts between 40000 and 50000 from branch name 'CIDCO'.

b) Write a procedure using cursor add an interest of 3% to the balance of all accounts having balance > 5000.

2. Using bus database

a) Write a procedure using cursor to display the details of a driver,

(Accept driver name as input parameter).

b) Write a cursor to display the details of the buses that run on route_no = 1 and route_no = 2 . (Use two different cursors for route_no = 1 and route_no = 2).

3. Using student-competition database

a) Write a procedure using cursor which will list all the competitions in which students studing in the 5th std have won 1st prize in 1995.

b)Write a procedure using cursor to give competition wise 1st or 2nd rank holders for all the competitions held in the year 2001

SET B

1) Using client-policy database

a) Write a procedure that uses a cursor to update the type_of_premium field of the relationship table using the function already created .

'yearly' for 'y'.
'half yearly' for 'h'.
'quarterly' for 'q'.

b) Write a procedure using cursor to transfer all the clients of agent_id = 10 to agent_id = 15. Delete all records of agent_id 10 from all the tables.

2) Using real estate database

a) Write a procedure which will calculate the total commission earned by each agent.
b) Write a procedure which will increase the commission by 5% of all agents who have sold 2bhk flats .

3) Using mobile billing database

a) Write a procedure which will give 10% discount on the final bill amt for the cycle period 'oct-nov' if the finalbillamt>700;
b) Write a procedure which will give the details of bill for the specific cycleperiod. (accept cycleperiod as a input parameter). Bill details contains bill_no, customer name, total no. of calls, total talktime , final bill amount.

SET C

1) Using railway reservation database

a) Write a cursor to find the confirmed bookings of all the trains on 18-05-2009
b) Write a cursor to find the total number of berths not reserved for all the trains on 18-05-2009.

2) Using Sales Management Database

a) Create a procedure that will increase the product cost by Rs. 1000 in productmaster table based on the following conditions (using cursor)

A product must be sold at least once.
A product sale price must be more than 1600

b) Create a new table called emp_performance having fields employeeID , total sale and performance_level , Set the performance levels of the employees in the tables based on the given range.

Sale range	Performance
> 8000	Excellent
> 5000 and < 8000	Good
> 3000 and < 5000	Average
< 3000	Poor

The performance level will be judged on the basis of the total transaction amount per employee.

Assignment Evaluation

0: Not Done [ ]	1:Incomplete [ ]	2.Late Complete [ ]
3:Needs Improvement [ ]	4:Complete [ ]	5:Well Done [ ]

Signature of the Instructor Date of Completion ____/____/_____

End of Session

Session 5: Triggers

Start Date __/_____/_____

Objectives

To learn about:

- Triggers and methods of creating / deleting triggers.

Reading

- You should know the following topics before starting this exercise.
- How to create databases, tables in MSQL and need to have completed the stored procedures tutorials
- You should know the following topics before starting this exercise.
  How to create databases, tables in MSQL and need to have completed the stored procedures tutorials

Ready References

Triggers are stored routines much like procedures and functions however they are attached to tables and are fired automatically when a set action is performed on the data in that table.
A trigger is defined to activate when an INSERT, DELETE, or UPDATE statement executes for the associated table. A trigger can be set to activate either before or after the triggering statement. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.

Trigger Types

A trigger is a stored unit of code that is attached to a table within the database. A trigger cannot be called or used in a select statement in the same way a procedure or function can but are called automatically when an action takes place on the table it is associated with. There are 12 different types of triggers that are possible and these can be split into 3 different groups, they all have the same functionality and limitations but are called at different stages of a table action.
It may be a good point to tell you that it’s only possible to have one trigger per type in MySQL so for example if we have a table called emps it would only be possible to have one BEFORE INSERT trigger against it.
Unlike procedures or functions we cannot invoke triggers manually, they must be called as a result of the relevant table action. To fire the trigger a record needs to be inserted into the relevant table.

```
CREATE TRIGGER trigger_name trigger_time trigger_event ON tbl_name FOR EACH
ROW trigger_stmt
```

Use the CREATE TRIGGER keywords to tell MySQL that we are creating a trigger, then name the trigger using standard MySQL naming conventions. Its best to keep the name short but descriptive, its also a good idea to include a code or naming convention which shows the type of trigger that is being created,

Trigger Time

The trigger time is at what stage the process takes place. This can be either BEFORE or AFTER. If using before this means that the trigger code will be able to work with both the values currently stored in there original state and the new values. If you use after then this is working on the table after it has been updated. This is important in an environment which uses transactions there may be situations where we don't want to fire the trigger until a commit has taken place.

Trigger Event

The trigger event is the event that happens on the table that causes the trigger code to fire. This can be one of the following 3 types.

INSERT / UPDATE / DELETE

If you want the trigger to fire when an insert on the table takes place the you need to use the INSERT keyword, if you want the trigger to fire when an update takes place then use UPDATE and of course if you want the action to fire when a delete takes place use the DELETE keyword.

We can start to see the different types of triggers it's possible to create using a combination of different trigger events and times. So for example it's possible to have both a BEFORE INSERT and AFTER INSERT trigger type. It's this combination with the use of a keyword that creates the 12 different types of trigger we can use.

Table Name

The table name is then specified using the ON table_name keywords. It may be a good point to tell you that it's only possible to have one trigger per type in MySQL so for example if we have a table called emps it would only be possible to have one BEFORE INSERT trigger against it. This is unlike some other database implementations such as Oracle which allows multiple triggers for any given trigger type.

FOR EACH ROW

The last part of the trigger definition before the actual code is the FOR EACH ROW keyword, this is the only part of the trigger specification which is not mandatory. If it is used then the trigger will fire once for all records that are being worked on. If it is not specified the trigger will fire once only regardless of the number of records being updated.

Finally we can implement the code which will be fired when the trigger is called. As with procedures and functions this can be a single line directly after the trigger specification or multiple lines of code contained within a BEGIN and END. The code can contain any valid stored procedure statements, this includes loops IF, LOOP, CASE and as from version 5.0.10 SQL statements.

Creating Triggers

To create a trigger the code needs to be entered into the MySQL command line either by typing it in long hand or by using a source file.

```
mysql> drop database if exists pers;
mysql> create database pers;
mysql> use pers;

mysql> create table emps(emp_id int NOT NULL,
          emp_name varchar(30),
          dept_id int,
          dept_name varchar(30),
          salary decimal(7,2),
          primary key(emp_id));

mysql> insert into emps (emp_id,emp_name,dept_id,salary)
         values (1,'Roger',1,2000.00),(2,'John',2,2500.00),(3,'Alan',1,2100.00);

mysql> select * from emps;

mysql> create table dept (dept_id int NOT NULL,
          description varchar(30),
          primary key(dept_id));

mysql> insert into dept (dept_id,description)
         values (1,'Information Technology'),(2,'Sales');

mysql> select * from dept;
```

The first trigger will be a simple one line trigger to demonstrate the basics of trigger creation. Our clients are a group of parallel universe accountants and when ever they enter any names they do so in reverse order. We need to create a simple trigger so that on entry of names into our emps table the value is reversed.
So before we create the trigger there are a number of decisions we need to take. First what to call our trigger we want to update each emp_name column of our emps table when a record is inserted. We also want to change the values so we need to fire the trigger before the action is completed. So using this information we can see we want a BEFORE INSERT ON emps FOR EACH ROW trigger. The specification for this would be as follows.

```
CREATE TRIGGER bi_emps_fer BEFORE INSERT OF emps FOR EACH ROW...
```

OLD and NEW
Rather than updating the table by name we can use the OLD and NEW keywords to access the data in the record we are dealing with. OLD is used for the values of the column before the change was made and NEW holds the value of the column after the column has changed. Therefore in our program we can access the emp_name column using new.emp_name

```
mysql> create trigger bi_emps_fer before insert on emps for each row
set new.emp_name := reverse(new.emp_name);
```

This is a simple one line trigger so we do not need to enclose the code within a BEGIN and END. Unlike procedures or functions we cannot invoke triggers manually, they must be called as a result of the relevant table action. To fire the trigger a record needs to be inserted into the relevant table.

```
insert into emps (emp_id,emp_name) values (4,'Dave');
Query OK, 1 row affected (0.03 sec)
```

At this stage there is no visual clue that the trigger has actually fired. Selecting against the table will show use if the trigger has fire and reversed the emp_name column.

```
mysql> select * from emps where emp_id = 4;

+ -------- ---------- --------- ----------- -------- --------------------
I emp_id I emp_name I dept_id I dept_name I salary I
 -------- ---------- --------- ----------- ------------------------------
I    4    I evaD       I   NULL I NULL       I  NULL I
 -------- ---------- --------- ----------- -------- ----------------------
1 row in set (0.02 sec)
```

This shows us that the column was indeed reversed so the trigger must have fired.

It is also possible, again like functions and procedures, to create more complex triggers which take multiple lines of code to implement. To do this the BEGIN and END keywords are use to tell the compiler where the trigger code starts and ends. To write longer sections of code a semi colon ; is used to terminate the lines therefore an alternative delimiter needs to be set. A common delimiter to use in this instance is a double slash like so / /, this is set in MySQL using the following syntax.

```
Mysql> delimiter / /
```

Once the delimiter has been set the standard MySQL procedural code can be used to build the trigger body.

You have a company that wants to pay its employees based on the number of characters in their name.

This trigger will contain more than one line of code so we therefore need to contain the code within a begin and end to let the compiler know where the statements begin and end. There are a number of ways we could work out the number of characters in the employee's name but in this instance the standard MySQL function LENGTH will be used.

```
mysql> delimiter / /
mysql> create trigger bi_emps_fer before insert on emps for each row
begin
  declare namelength numeric;
  set namelength = length(new.emp_name);
  set new.salary = new.salary * namelength;
end / /

mysql> delimiter ;
```

In the following example we are modifying the salary of Employee table before updating the record of the same table. Example :

```
mysql> delimiter //
mysql> CREATE TRIGGER updtrigger BEFORE UPDATE ON Employee
   -> FOR EACH ROW
   -> BEGIN
   -> IF NEW.Salary<=500 THEN
   -> SET NEW.Salary=10000;
   -> ELSEIF NEW.Salary>500 THEN
   -> SET NEW.Salary=15000;
   -> END IF;
   -> END
   -> //
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;

mysql> UPDATE Employee
       -> SET Salary=500;
Query OK, 5 rows affected (0.04 sec)
Rows matched: 7  Changed: 5  Warnings: 0

mysql> SELECT * FROM Employee;
+-----+-------------+------------+-------------------------+-----------+--------+
| Eid | Ename    | City      | Designation          | Salary | Perks |
+-----+-------------+------------+--------------------------+----------+---------+
| 1    | Rahul     | Delhi     | Manager              | 10000 | 853    |
| 2    | Gaurav   | Mumbai | Assistant Manager  | 10000 | 853    |
| 3    | Chandan | Banglore| Team Leader        | 10000 | 999    |
| 5    | Tapan    | Pune     | Developer           | 10000 | 1111 |
| 6    | Amar     | Chennai | Developer           | 10000 | 1124  |
| 7    | Santosh | Delhi     | Designer             | 10000 | 865    |
| 8    | Suman   | Pune     | Web Designer       | 10000 | 658    |
+-----+--------------+-----------+--------------------------+----------+---------+
7 rows in set (0.00 sec)
```

Write a function to give the total sale of the employee in the month of march in year 2009. (pass emp_id , month and year parameters to the function and return the total sale.)

```
mysql> UPDATE Employee
   -> SET Salary=1500;
Query OK, 7 rows affected (0.03 sec)
Rows matched: 7  Changed: 7  Warnings: 0

mysql> SELECT * FROM Employee;
mysql> SELECT * FROM Employee;
+-----+-------------+------------+-------------------------+-----------+--------+
| Eid | Ename    | City      | Designation          | Salary | Perks |
+-----+------------+-----------+--------------------------+----------+---------+
| 1    | Rahul     | Delhi     | Manager              | 15000 | 853    |
| 2    | Gaurav   | Mumbai | Assistant Manager   | 15000 | 853   |
| 3    | Chandan | Banglore| Team Leader         | 15000 | 999   |
| 5    | Tapan    | Pune    | Developer           | 15000 | 1111 |
| 6    | Amar     | Chennai | Developer           | 15000 | 1124  |
| 7    | Santosh  | Delhi    | Designer            | 15000 | 865   |
| 8    | Suman    | Pune    | Web Designer        | 15000 | 658   |
+-----+--------------+------------+--------------------------+----------+---------+
7 rows in set (0.00 sec)
```

DROP TRIGGER
The general syntax of DROP TRIGGER is :

```
    DROP TRIGGER trigger_name
```

This statement is used to drop a trigger. Example of Dropping the Trigger :

```
mysql> DROP TRIGGER updtrigger;
Query OK, 0 rows affected (0.02 sec)
```

Validating Data with Triggers

A typical and traditional use of triggers in relational databases is to validate data or implement business rules to ensure that the data in the database is logically consistent. These triggers can be referred to as check constraint triggers.
Such a trigger should typically prevent a DML operation from completing if it would result in some kind of validation check failing.
For e.g. we would like to add a trigger before insert / update on account table which checks that only those insertions / updations are allowed where the amount value is > 0.

Unfortunately there is no such validation facility available in MySQL 5.0 and 5.1
But we can use a workaround to force a trigger to fail in such a way that it prevents the DML statement from completing and provides a marginally acceptable error message.

Following is an example of a trigger that ensures that there will be no negative account balance. If a negative account balance is detected, the trigger attempts to execute a SELECT statement that references a nonexistent column. The name of the column includes the error message that we will report to the calling program.

```
CREATE TRIGGER acc_balance_bu
   BEFORE UPDATE ON account
   FOR EACH ROW
   BEGIN
       DECLARE dummy INT;
  IF NEW.balance < 0 THEN
              SELECT ‘Account balance can not be less than 0’ into dummy
              FROM account  Where account_id = NEW.account_id;
   END IF;
END;
```

Now lets see how the above trigger works. Consider the following statements

```
mysql->  SELECT * from account where account_ id = 1;
 +------------------+---------------+
  | account_id    |    balance  |
 +------------------+---------------+
  |           1     |    800.00    |
 +------------------+---------------+
```

```
mysql->UPDATE account SET balance = balance - 1000 WHERE account_id = 1;
```

```
ERROR  1054 (42522) : Unknown column 'Account balance can not be less than 0' in 'field
list'
```

```
mysql->  SELECT * from account where account_ id = 1;

 +-----------------+---------------+
  | account_id   |    balance  |
 +-----------------+---------------+
  |           1    |   800.00    |
 +-----------------+---------------+
```

The above example shows that the trigger prevented the update statement from proceeding. Although the error code is not ideal, and the error message is embedded in another error message, we at least have prevented the update statement from creating a negative balance, and we have provided an error message that does include the reason why the UPDATE was rejected.

Other method to handle the situation is to create the error table as follows

```
CREATE TABLE 'Error' (
        'ErrorGID' int(10) unsigned NOT NULL auto_increment,
        'Message' varchar(128) default NULL,
        'Created' timestamp NOT NULL default
CURRENT_TIMESTAMP
        on update CURRENT_TIMESTAMP,
        PRIMARY KEY ('ErrorGID'),
        UNIQUE KEY 'MessageIndex' ('Message'))
        ENGINE=MEMORY
        DEFAULT CHARSET=latin1
        ROW_FORMAT=FIXED
        COMMENT='The Fail() procedure writes to this table
        twice to force a constraint failure.'
```

The important things to note about this table are the unique key in the Message column, this allows us to easily create the error and the fact the table uses the MEMORY engine. This means the table is stored in memory and not in the file system, using that we don't need to clear the table down or worry about the table becoming full.

Next we need to create a procedure to insert into this table to raise the unique key violation, this stage isn't strictly necessary but makes the trigger code we will produce in a moment much more readable. It also means we can consistently call the same routine in any number of triggers (or even in Functions and Procedures).

```
DELIMITER $$
DROP PROCEDURE IF EXISTS ‘Fail‘$$
CREATE PROCEDURE ‘Fail‘(_Message VARCHAR(128))
BEGIN
  INSERT INTO Error (Message) VALUES (_Message);
INSERT INTO Error (Message) VALUES (_Message);
END$$
DELIMITER;
```

As you can see the procedure accepts a parameter called _Message, this is then used as the Message column value in the inserts. By calling the same insert statement twice the unique_key constraint is violated on the error table, because MySQL reports the column value in the error message this _Message parameter is added to the error message. We can call this procedure from the command line to see this in action.

```
mysql> call fail(’Salary must be over 10’);
ERROR 1062 (23000): Duplicate entry ’Salary must be over 10’ for key 2
```

There are two things to note here, firstly we get our error message returned which is great as it means the user can clearly see why the error was raised. The second thing you may notice is that the error number and description point to a duplicate entry error, this isn’t ideal but for now is the only way we can raise the error.

We now have the table and a procedure capable of raising an error safely without effecting any other part of the database. That’s one of the key things here, we could raise the error without using a specific table or the procedure but doing so won’t effect any other part of the database.
We now need to add this to a trigger so that the error is raised when the constraint fails.
We will use the salary example we mentioned earlier. I you have been using the pers schema you will have a table called emps, this table has a column called salary. Let’s create a trigger which will enforce a constraint on that column so that the value cannot be less than 10 and not higher than 100.

```
DELIMITER $$
create trigger salary_check before insert on pers.emps for each row
begin
 if new.salary < 10 or new.salary > 100 then
      call fail(’Salary not in allowed range’);
 end if;
end $$
DELIMITER;
```

The trigger simply checks the value of the new.salary column, if it’s not within the parameters allowed a call to the fail routine is called. This will then stop the record being inserted into the table. Let’s look at that in action, first we can take a look to see what values are in our emps table.

```
mysql> select * from emps;
 -------- ---------- --------- -------- --------
| emp_id | emp_name | dept_id | salary | bonus  |
 -------- ---------- --------- -------- --------
|      0 | Barry    |    NULL |   0.00 |   NULL |
|      1 | Paul     |       1 | 100.00 | 100.00 |
|      2 | John     |       2 | 200.00 | 100.00 |
|      3 | Alan     |       1 | 300.00 | 100.00 |
 -------- ---------- --------- -------- --------
4 rows in set (0.00 sec)
```

We can first test that records that have a valid salary are inserted correctly.

```
mysql> insert into emps (emp_id,emp_name,dept_id,salary)
    -> values (4,'Sally',1,90.00);
Query OK, 1 row affected (0.06 sec)

mysql> select * from emps;
 -------- ---------- --------- -------- --------
| emp_id | emp_name | dept_id | salary | bonus  |
 -------- ---------- --------- -------- --------
|      0 | Barry    |    NULL |   0.00 |   NULL |
|      1 | Paul     |       1 | 100.00 | 100.00 |
|      2 | John     |       2 | 200.00 | 100.00 |
|      3 | Alan     |       1 | 300.00 | 100.00 |
|      4 | Sally    |       1 |  90.00 |   NULL |
 -------- ---------- --------- -------- --------
5 rows in set (0.00 sec)
```

The record was inserted into the table without a problem, now we can try and insert a record which breaks the constraint and see what happens.

```
mysql> insert into emps (emp_id,emp_name,dept_id,salary)
    -> values (5,'Penny',1,200.00);
ERROR 1062 (23000): Duplicate entry 'Salary not in allowed range' for key 2.
```

This time the value wasn't in the range we have specified and the trigger raised the error just as we wanted.

Self Activity

Execute the above example of triggers.

Practical Assignment

SET A

1. Using Bank Database

a) Write a trigger which will execute when account_no is less than 0 . Display the appropriate message.
b) Write a trigger which will execute when loan_amount is updated. Do not allow to update. Display the message that ' loan amount once given cannot be updated."

2. Using Bus Transportation database

a) Define a trigger after insert or update the record of driver if the age is between 18 and 50 give the message "valid entry" otherwise give appropriate message.
b)Define a trigger after delete the record of bus having capacity < 10. Display the message accordingly.

3. Using Student Competition database

a) Write a trigger before insertion on the relationship table. if the year entered is greater than current year, it should be changed to current year.
b) Create a new table 'tot_prize' containing the fields stud_reg_no and no_of_prizes.
Write a trigger after insert into the relationship table between student and Competition. It should increment the no_of_prizes in the table 'tot_prize' for the NEW stud_reg_no by 1.

SET B

1. Using Client - policy database :

a) Write a trigger on client to implement age as a derived attribute. Birth_date should be less than or equal to the current_date . If it is not, raise error else calculate client age from the birth_date. If age is in the given range of the selected policy(min_age_limit & max_age_limit) , then update the age field else raise error.

b) Write a trigger on client table after insert to update the term in the relationship table as
Term = maturity_age of the policy the client has taken – age of the client.
c) Write a trigger on relationship table before insert to implement check constraint on type_of_premium field. The type should be one of the ('h','y','q').

2. Using Real Estate Agency Database

a) First, alter table estate by adding a column status varchar(20).
(Initially all status are null) Write a trigger which will fire after insertion on transition table which will change the status of the estate as 'sold'.

b) Write a trigger which will fire after any estate is deleted from the transition table it will also deletes from the estate table.

3. Using Mobile Billing database

a) Calculate the length of mobileno. Write a trigger which will fire before insert on the customer table which check that the mobileno must be of 10 digit. If it is more or less then it display the appropriate message.
b) Write a trigger which will fire after any customer is deleted from the customer table then that customer must be deleted from custcallinfo and bil.

SET C

1. Using Railway Reservation Database

a)create a trigger to validate train arrival time must be less than train departure time.
b) Write a trigger which will be activated before changing the status field in the ticket table

2. Using Sales Management Database

a) Write a trigger to update the quantity field in product_master table as quantity of the product(in product_master) - quantity of the product ordered(in transaction_product). Allow updation if quantity inserted in the transaction_product table is greater than zero. Else give the appropriate message.
b) Write a trigger to implement the check constraint on transaction table to check that transaction_date <= delivery_date and transaction_date <= current_date .

Assignment Evaluation

0: Not Done [ ]	1:Incomplete [ ]	2.Late Complete [ ]
3:Needs Improvement [ ]	4:Complete [ ]	5:Well Done [ ]

Signature of the Instructor Date of Completion ____/____/_____

End of Session

Session 6: Views

Start Date __/_____/_____

Objectives

To learn about:

- views and methods of creating / deleting views.

Reading

- You should know the following topics before starting this exercise.
- How to create databases, tables in MSQL , writing of simple and nested queries.

Ready References

Views are simply a method of storing an SQL statement as a persistent part of the database. Prior to views if you wished to retrieve only a sub set of data from a table or data from two or more tables this needed to be done using an sql statement which once called would need to be rewritten if the same data was needed again.

Views are effectively stored SQL statements. Therefore to use views we will need to create some test tables and data to create views against, run the following script into MySQL to produce the test database we will be using in these first examples.

```
mysql->drop database if exists pers;

mysql->create database pers;

mysql->use pers;

mysql->create table emps(emp_id int NOT NULL,
          emp_name varchar(30),
          dept_id int,
          salary decimal(7,2),
          primary key(emp_id));

mysql->insert into emps (emp_id,emp_name,dept_id,salary)
         values (1,'Roger',1,2000.00),(2,'John',2,2500.00),(3,'Alan',1,2100.00);
```

Creating Simple Views

The syntax to create a view is fairly simple.

```
CREATE VIEW view_name AS SELECT column_list... FROM table_name;
```

CREATE VIEW tells MySQL that a view is being created, a name is given to the view, this can be any name which follows the naming conventions for MySQL database objects but it should be noted that as views are dealt with in a similar way to tables view names must not be the same as a table name in the same database. Finally a select statement is specified, this can be almost any valid SQL statement using a single or multiple tables, there are some restrictions, but for the most part all standard SQL statements will work.

The most simple form of a view is a simply select * from a table.

```
mysql->create view v_emp as select * from emps;
Query OK, 0 rows affected (0.00 sec)

mysql->select * from v_emp;

+--------+----------+---------+-------------+
| emp_id | emp_name | dept_id | salary   |
+--------+----------+---------+-------------+
|    1   | Roger     |     1  | 2000.00 |
|    2   | John      |     2        | 2500.00 |
|    3   | Alan      |     1  | 2100.00 |
+---------+-------------+--------+----------+
3 rows in set (0.00 sec)
```

This is effect an exact replica of the SQL used to query the table itself. One very important point to make note of at this early stage is that the view creation makes no copy of the data, it’s simply storing the SQL. We defined the statement with * which as you may be aware equates to all of the columns in the table, but just as with a table itself we can now specify particular columns to be returned from the view like so.

```
mysql->select emp_name from v_emp;

+------------+
| emp_name |
+------------+
| Roger      |
| John       |
| Alan       |
+------------+
3 rows in set (0.00 sec)
```

We have seen that we can create a view based on the whole table, but its also possible to create a view on a subset of the data contained in a table. Let’s say we have a combined HR and Payroll system and we want a view on the emps table that omits the salary field so that our HR staff doesn’t see the employee’s salary column.

```
Create view v_hr_emp as select emp_id, emp_name, dept_id from emps;
Query OK, 0 rows affected (0.00 sec)

mysql->select * from v_hr_emp;
+--------+----------+------------+
| emp_id | emp_name | dept_id |
+--------+----------+---------+
|    1   | Roger   |     1    |
|    2   | John    |     2    |
|    3   | Jane    |     1    |
+--------+----------+---------+
3 rows in set (0.00 sec)
```

The users in the HR department can now be given access to the new view but not the underlying table. This allows them full access to run SQL statements against the view but protect the salary information.

Dropping views

The syntax to drop a view is as follows:

```
DROP VIEW [IF EXISTS] view_name, view_name .... [RESTRICT/CASCADE];
```

Simply use the key words DROP VIEW and the view or views you wish to drop. The keywords RESTRICT and CASCADE will be parsed successfully but are not currently supported so for all intents and purposes can be ignored. You can add IF EXISTS after the DROP VIEW to avoid any error messages being displayed if the view currently does not exist.

Updatable and Insertable Views

Some views are updatable. That is, you can use them in statements such as UPDATE, DELETE, or INSERT to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view non-updatable. To be more specific, a view is not updatable if it contains any of the following:

1) Aggregate functions (SUM(), MIN(), MAX(), COUNT(), and so forth)
2) DISTINCT
3) GROUP BY
4) HAVING
5) UNION or UNION ALL
6) Sub query in the select list
7) Certain joins
8) Non-updatable view in the FROM clause
9) A subquery in the WHERE clause that refers to a table in the FROM clause
10) Refers only to literal values (in this case, there is no underlying table to update)
11) Use of a temporary table always makes a view non-updatable)
12) Multiple references to any column of a base table.

With respect to insert ability (being updatable with INSERT statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

There must be no duplicate view column names.
The view must contain all columns in the base table that do not have a default value.
The view columns must be simple column references and not derived columns.

A view that has a mix of simple column references and derived columns is not insertable, but it can be updatable if you update only those columns that are not derived. Consider this view:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because col2 is derived from an expression. But it is updatable if the update does not try to update col2. This update is allowable:

UPDATE v SET col1 = 0;
This update is not allowable because it attempts to update a derived column:

UPDATE v SET col2 = 0;

It is sometimes possible for a multiple-table view to be updatable. For this to work, the view must use an inner join (not an outer join or a UNION). Also, only a single table in the view definition can be updated, so the SET clause must name only columns from one of the tables in the view. Views that use UNION ALL are disallowed even though they might be theoretically updatable, because the implementation uses temporary tables to process them.

For a multiple-table updatable view, INSERT can work if it inserts into a single table. DELETE is not supported.

Practical Assignment

SET A

1. Using the Bus – Driver database

1. Create a view which contains details of bus no 10 along with details of all drivers who have driven that bus.
2. Create a view which contains details of all the routes which are between the stations 'kothrud depo' and 'corporation'

2. Using the student - competition database

1. Create a view over the competition table which contains only competition name and its type and it should be sorted on type.
2. Create a view containing student name, class, competition name, rank and year. the list should be sorted by student name.

3. Using the Bank database

1. Create a view which contains all the customer details along with the details of all accounts of that customer.
2. Create a view which contains details of all loans from the 'sadashiv peth' branch.

SET B

1. Using the client - policy database
1. Create a view to list the details of all clients from branch office ' Khadki'.
2. Create a view to list the details of all clients whose policy_date is '______'.

2. Using the Real estate - Agent database

1. Create a view of all real estates which are in 'Pimpri' area and which are not sold.
2. Create a view containing the details of all the estates which are sold by the agent 'Mr. Kale' and also the details of customers who have bought the estates.

3. Using the Mobile- billing database

1. Create a view containing details of all the bills generated for the customer 'Mrs. Joshi'.
2. Create a view containing details of all customers who have registered under the '———' plan.

SET C

1. Using the Sales Management database

1. Create a view containing details of all sales done by employee ———'.
2. Create a view to list the details of all customers who have taken 'fridge ' .

2. Using the Railway - reservation database

1. Create a view containing the details of all the passengers who have booked a ticket for the 'geetanjali express' on 3/6/09
2. Create a view to list the passenger names whose ticket status is 'Waiting' of 'Shatabdi express' on date ————.

Assignment Evaluation

0: Not Done [ ]	1:Incomplete [ ]	2.Late Complete [ ]
3:Needs Improvement [ ]	4:Complete [ ]	5:Well Done [ ]

Signature of the Instructor Date of Completion ____/——/———

End of Session

References

1. Fundamentals of Database Systems (4th Ed) By: Elmasri and Navathe
2. Database System Concepts (4th Ed) By: Korth, Sudarshan, Silberschatz
3. MySQL The Complete Reference By Vikram Vaswani
4. Learning MySQL by O'reilly
5. MySQL in Nut Shell by Dyer 2nd Edition
6. MySQL 5 for professionals by Ivan Bayross and Sharanam Shah (SPD)
7. Various ebooks from internet and help from internet
8. Website for MySQL http: / / dev.mysql.com

****