Lab Manual
for
CSE-302 (Database System Lab)

Credit hour: 1.5, Contact hour: 3 hrs. Per week

Department of Computer Science & Engineering

Green University of Bangladesh

Dhaka, Bangladesh

Green University of Bangladesh (GUB)
Dept. of Computer Science and Engineering

Green University of Bangladesh

Department of Computer Science & Engineering

CSE-302
Database System Laboratory

Student ID	
Student Name	
Section	
Name of the Program	**BSc. in CSE**
Name of the Department	**Computer Science and Engineering**

INDEX

INSTRUCTIONS FOR LABORATORY

- The experiments are designed to illustrate about different syntax of database management system and its operations. Conduct the experiments with interest and an attitude of learning.
- Students should come with thorough preparation for the experiment to be conducted.
- Students should come with proper dress code.
- Students will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.
- Work quietly and carefully (the whole purpose of experimentation is to solve different database related problem using SQL query) and equally share the work with your partners.
- Be honest in developing and representing your SQL program. If a particular program output appears wrong repeat the program carefully.
- All presentations of relational database, outputs and key constraints should be neatly and carefully done.
- If you finish early, spend the remaining time to complete the laboratory report writing.
- Handle instruments with care. Report any breakage or faulty equipment to the Instructor. Shutdown your computer you have used for the purpose of your experiment before leaving the Laboratory.

Green University of Bangladesh (GUB)
Dept. of Computer Science and Engineering

COURSE OUTLINE

1	**School**	Faculty of Science and Engineering (FSE)
2	**Department**	Computer Science and Engineering
3	**Programme**	B.Sc in Computer Science and Engineering
4	**Name of Course**	Database System Lab
5	**Course Code**	CSE 302
6	**Trimester**	
7	**Pre-requisites**	None
8	**Status**	System Courses
9	**Credit Hours**	1.5
10	**Section**	
11	**Class Hours**	TBA
12	**Class Location**	TBA
13	**Course website**	TBA
14	**Instructor**	TBA
15	**Contact**	TBA
16	**Office**	
17	**Counselling Hours**	see table below
18	**Text Book**	1. Van Der Lans, R. F. (2007). SQL for MySQL Developers: a comprehensive tutorial and reference. Pearson Education.
19	**Reference**	1. http://www.mysqltutorial.org/ 2. https://www.w3schools.com/sql/sql_ref_mysql.asp 3. https://www.w3schools.com/sql/sql_ref_oracle.asp 4. Loney, K. (2004). Oracle database 10g: the complete reference. London: McGraw-Hill/Osborne. 5. Silberschatz, A., Korth, H. F., & Sudarshan, S. (1997). Database system concepts (Vol. 4). New York: McGraw-Hill.
20	**Equipment & Aids**	Bring your notebook. Xampp, Mysql and Heidisql software are installed in the respective laboratory computers. Do collect the software's for home practice.

Day	**Counseling Hours**	**Venue**

21	**Course Rationale**	MySQL is a free, open-source database management system (DBMS for short). A DBMS is a system that manages databases and connects them to software. For example, a MySQL database can be used to run a website, to run the database of an ERP or any other software. MySQL is a powerful, free open-source database management system that has been around for years. It is very stable and has a big community that helps maintain, debug and upgrade it. MySQL might not be as popular for larger systems that will mostly run on Microsoft SQL Server or Oracle. These proprietary DBMS are more scalable, have more resources available on the market and have more advanced features that MySQL.
22	**Course Description**	Concepts of database systems, Integrity constraint, DDL, DML, DTL, Introduction to SQL, Syntax, Aggregation function, relational operators, logical operators, string operations, Join functions; Query Processing, Hashing and Indexing, Query Optimization; Database Triggers- Row level triggers based on update, insert, delete; basic of data mining and data warehousing, PL/SQL, functions, sequences, procedures.
23	**Course Outcomes (CO)**	After completing this course students will be able to- **CO1:** Apply the basic knowledge of database management system to solve data storage and querying issues. **CO2:** Generate SQL query to retrieve information from database. **CO3:** Design the relational database as per user requirements with modern tools and techniques.
24	**Teaching Methods**	Lecture, Laboratory experiments, Project developments.
25	**Topic Outline**	

Class	Topics or Assignments	COs	Reading Reference	Activities
1	Introduction to database and MySQL	1	Lab. Manual, Experiment No. 1	Laboratory Experiment
2	Managing MySQL databases and tables in MySQL	2	Lab. Manual, Experiment No. 2	Laboratory Experiment
3	Implementation of Integrity constraints in MySQL	2	Lab. Manual, Experiment No. 3	Laboratory Experiment
4	Modifying MySQL databases and Updating Data in MySQL Table	2	Lab. Manual, Experiment No. 4	Laboratory Experiment
5	Querying and Filtering data in MySQL Table (Extended)	2-3	Lab. Manual, Experiment No.5	Laboratory Experiment
6-7	Implementation of MySQL Aggregate Function, Joining	2-3	Lab. Manual, Experiment No. 6	Laboratory Experiment
8	Implementation of Databases Triggers	2-3	Lab. Manual, Experiment No. 7	Laboratory Experiment
9	Implementation of Databases Transactions and Multiuser Usage	2-3	Lab. Manual, Experiment No. 8	Laboratory Experiment

	10	Final Term Examination	1-2-3		
	11	Project Presentation, Report Submission.			

26 Assessment and Marks Distribution:

Students will be assessed on the basis of their overall performance in all the exams, quizzes, and class participation. Final numeric reward will be the compilation of (tentative):

- ❖ Attendance and Performance (AP) (10%)
- ❖ Capstone Project Presentation (25%)
- ❖ Lab Test (LT) (25%)
- ❖ Lap Report (LR) (10%)
- ❖ Lab Final (LF) (30%)

27 Assessment Methods of COs

Assessment methods of COs are given below:

Assessment						
COs	**AP**	**LT**	**LR**	**LF**	**Assignment**	Capstone Project
CO1				√		
CO2		√	√	√		√
CO3		√	√	√		√

28 Mapping of COs with PLOs

Mapping of COs with program outcomes (POs) are given below:

Program Outcomes (PLOs)												
COs	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	√											
CO2		√										
CO3				√								

29 Grading Policy

The following chart will be followed for grading. This has been customized from the guideline provided by the School of Engineering and Computer Science.

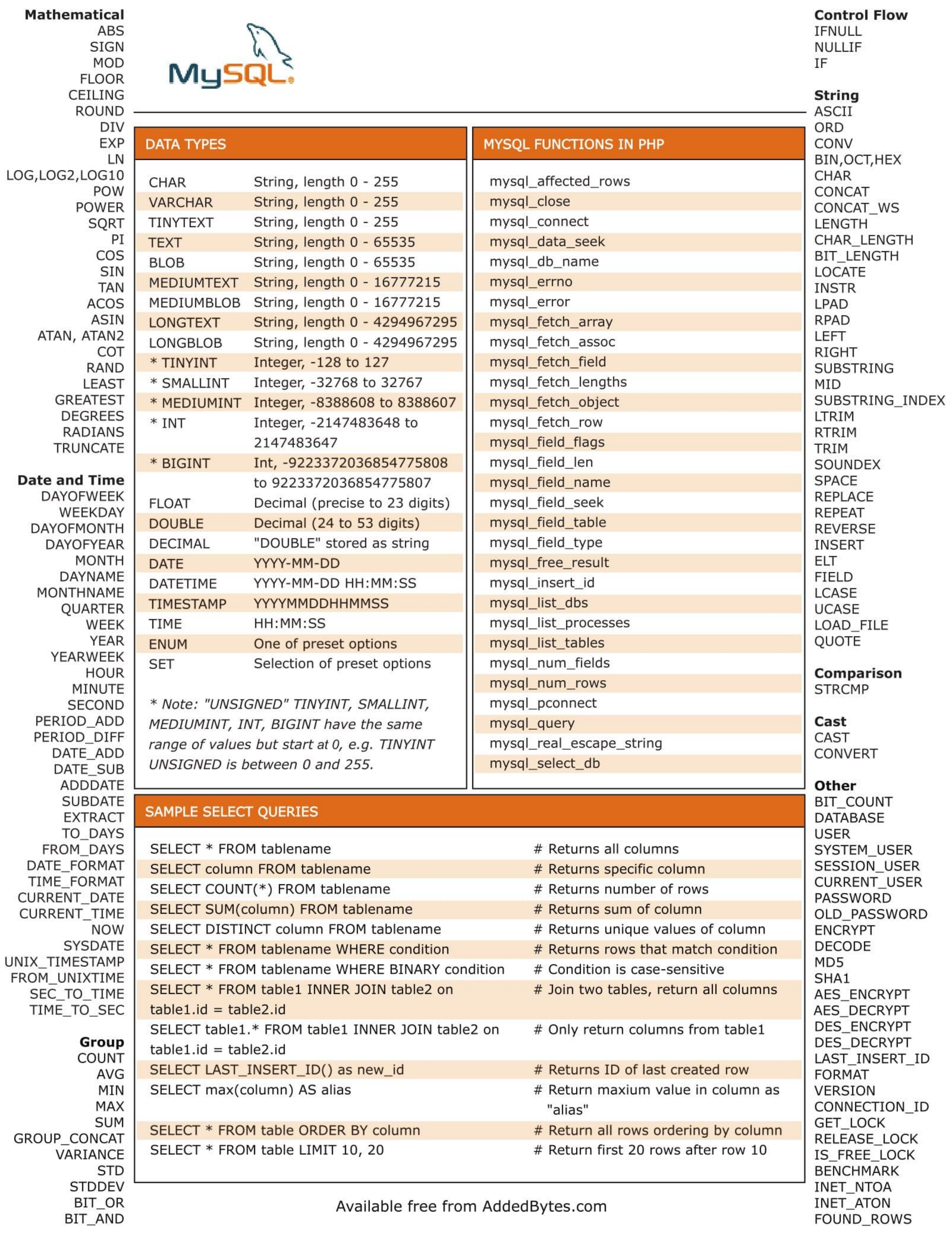
A+	**A**	**A-**	**B+**	**B**	**B-**	**C+**	**C**	**D**	**F**
80 and above	75-<80	70-<75	65-<70	60-<65	55-<60	50-<55	45-<50	40-<45	<40

29 Additional Course Policies

1. 1. Lab Reports:
Report on previous Experiment must be submitted before the beginning of new experiment. A bonus may be obtained if a student submits a neat, clean and complete lab report.
2. 2. Examination:
There will be a mid-term exam and final exam both of which will be closed book.
3. 3. Unfair means policy:
In case of copying/plagiarism in any of the assessments, the students involved will receive zero marks. Zero Tolerance will be shown in this regard. In case of severe offences, actions will be taken as per university rule.
4. 4. Counseling:
Students are expected to follow the counseling hours posted. In case of emergency/unavoidable situations, students can e-mail me to make an appointment.
5. 5. Policy for Absence in Class/Exam:

		If a student is absent in the class for anything other than medical reasons, he/she will not receive attendance. If a student misses a class for genuine medical reasons, he/she must submit an application with the supporting documents (prescription/medical report). He/she will then have to follow the instructions given by the instructor for make-up. In case of absence in the mid/final exam for medical grounds, the student must also get his/her application forwarded by the head of the department before a make-up exam can be taken. It is recommended that the students inform the instructor beforehand through mail if they feel that they will miss a class/evaluation due to medical reasons.
30	**Additional Information**	a. Academic Calendar Fall 2018: http://www.green.edu.bd/academics/academic-calendar. b. Academic Information and Policies: http://www.green.edu.bd/academics/academic-rules-a-regulations. c. Grading and Performance Evaluation: http://www.green.edu.bd/academics/academic-rules-a-regulations. d. Proctorial Rules: http://www.green.edu.bd/administrator/proctors-office.

CHEATSHEET MYSQL DATABASE



Mathematical
ABS
SIGN
MOD
FLOOR
CEILING
ROUND
DIV
EXP
LN
LOG,LOG2,LOG10
POW
POWER
SQRT
PI
COS
SIN
TAN
ACOS
ASIN
ATAN, ATAN2
COT
RAND
LEAST
GREATEST
DEGREES
RADIANS
TRUNCATE

Date and Time
DAYOFWEEK
WEEKDAY
DAYOFMONTH
DAYOFYEAR
MONTH
DAYNAME
MONTHNAME
QUARTER
WEEK
YEAR
YEARWEEK
HOUR
MINUTE
SECOND
PERIOD_ADD
PERIOD_DIFF
DATE_ADD
DATE_SUB
ADDDATE
SUBDATE
EXTRACT
TO_DAYS
FROM_DAYS
DATE_FORMAT
TIME_FORMAT
CURRENT_DATE
CURRENT_TIME
NOW
SYSDATE
UNIX_TIMESTAMP
FROM_UNIXTIME
SEC_TO_TIME
TIME_TO_SEC

Group
COUNT
AVG
MIN
MAX
SUM
GROUP_CONCAT
VARIANCE
STD
STDDEV
BIT_OR
BIT_AND

MySQL

DATA TYPES

CHAR	String, length 0 - 255
VARCHAR	String, length 0 - 255
TINYTEXT	String, length 0 - 255
TEXT	String, length 0 - 65535
BLOB	String, length 0 - 65535
MEDIUMTEXT	String, length 0 - 16777215
MEDIUMBLOB	String, length 0 - 16777215
LONGTEXT	String, length 0 - 4294967295
LONGBLOB	String, length 0 - 4294967295
* TINYINT	Integer, -128 to 127
* SMALLINT	Integer, -32768 to 32767
* MEDIUMINT	Integer, -8388608 to 8388607
* INT	Integer, -2147483648 to 2147483647
* BIGINT	Int, -9223372036854775808 to 9223372036854775807
FLOAT	Decimal (precise to 23 digits)
DOUBLE	Decimal (24 to 53 digits)
DECIMAL	"DOUBLE" stored as string
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS
ENUM	One of preset options
SET	Selection of preset options

** Note: "UNSIGNED" TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT have the same range of values but start at 0, e.g. TINYINT UNSIGNED is between 0 and 255.*

MYSQL FUNCTIONS IN PHP

mysql_affected_rows
mysql_close
mysql_connect
mysql_data_seek
mysql_db_name
mysql_errno
mysql_error
mysql_fetch_array
mysql_fetch_assoc
mysql_fetch_field
mysql_fetch_lengths
mysql_fetch_object
mysql_fetch_row
mysql_field_flags
mysql_field_len
mysql_field_name
mysql_field_seek
mysql_field_table
mysql_field_type
mysql_free_result
mysql_insert_id
mysql_list_dbs
mysql_list_processes
mysql_list_tables
mysql_num_fields
mysql_num_rows
mysql_pconnect
mysql_query
mysql_real_escape_string
mysql_select_db

SAMPLE SELECT QUERIES

SELECT * FROM tablename	# Returns all columns
SELECT column FROM tablename	# Returns specific column
SELECT COUNT(*) FROM tablename	# Returns number of rows
SELECT SUM(column) FROM tablename	# Returns sum of column
SELECT DISTINCT column FROM tablename	# Returns unique values of column
SELECT * FROM tablename WHERE condition	# Returns rows that match condition
SELECT * FROM tablename WHERE BINARY condition	# Condition is case-sensitive
SELECT * FROM table1 INNER JOIN table2 on table1.id = table2.id	# Join two tables, return all columns
SELECT table1.* FROM table1 INNER JOIN table2 on table1.id = table2.id	# Only return columns from table1
SELECT LAST_INSERT_ID() as new_id	# Returns ID of last created row
SELECT max(column) AS alias	# Return maxium value in column as "alias"
SELECT * FROM table ORDER BY column	# Return all rows ordering by column
SELECT * FROM table LIMIT 10, 20	# Return first 20 rows after row 10

Available free from AddedBytes.com

Control Flow
IFNULL
NULLIF
IF

String
ASCII
ORD
CONV
BIN,OCT,HEX
CHAR
CONCAT
CONCAT_WS
LENGTH
CHAR_LENGTH
BIT_LENGTH
LOCATE
INSTR
LPAD
RPAD
LEFT
RIGHT
SUBSTRING
MID
SUBSTRING_INDEX
LTRIM
RTRIM
TRIM
SOUNDEX
SPACE
REPLACE
REPEAT
REVERSE
INSERT
ELT
FIELD
LCASE
UCASE
LOAD_FILE
QUOTE

Comparison
STRCMP

Cast
CAST
CONVERT

Other
BIT_COUNT
DATABASE
USER
SYSTEM_USER
SESSION_USER
CURRENT_USER
PASSWORD
OLD_PASSWORD
ENCRYPT
DECODE
MD5
SHA1
AES_ENCRYPT
AES_DECRYPT
DES_ENCRYPT
DES_DECRYPT
LAST_INSERT_ID
FORMAT
VERSION
CONNECTION_ID
GET_LOCK
RELEASE_LOCK
IS_FREE_LOCK
BENCHMARK
INET_NTOA
INET_ATON
FOUND_ROWS

Lab No.	Lab Tittle
01	**Introduction to database and MySQL**

Database, Database Server, and Database Language.

This section helps you get started with MySQL. We will start installing MySQL, downloading a sample database, and loading data into the MySQL server for practicing.

- Installing MySQL database server – show you step by step how to install MySQL database server on your computer.
- Downloading MySQL sample database – introduce you to a MySQL sample database named classicmodels. We also provide you links for downloading the sample database and its diagrams.
- Loading the sample database into your own MySQL database server – walk you through steps of how to load the classicmodels sample database into your MySQL database server for practicing.

Download MySQL Installer

If you want to install MySQL on Windows environment, using MySQL installer is the easiest way. MySQL installer provides you with an easy-to-use wizard that helps you to install MySQL with the following components:

- MySQL Server
- All Available Connectors
- MySQL Workbench with Sample Data Models
- MySQL Notifier
- Tools for Excel and Microsoft Visual Studio
- MySQL Sample Databases

To download MySQL installer, following link

http://dev.mysql.com/downloads/installer/.

https://www.apachefriends.org/download.html

Software Specification

xampp-win32-5.5.38-3-VC11-installer Download

Server:	MariaDB
Server version:	10.1.19-MariaDB mariadb.org binary distribution
Protocol version:	10
Connection:	127.0.0.1 via TCP/IP
TCP port:	3306

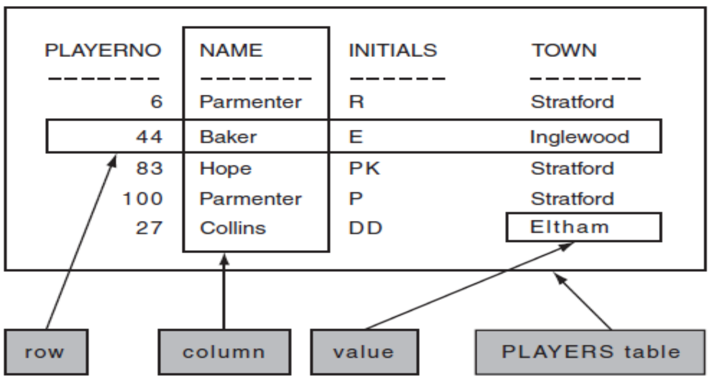
- A **database** consists of some collection of persistent data that is used by the application systems of some given enterprise and managed by a database- management system.
- A **database server** is a collection of programs that enables users to create and maintain a database. SQL (Structured Query Language) is a database language used for formulating statements processed by a database server.
- Commands are relayed to a database server with the help of special languages, called **database languages**. The relational database languages form one of these groups. An example of such a language is SQL.

The rest of this part concentrates on the following terms used in the relational model, which appear extensively in this book:

Table, Column, Row, Null value, Constraint or integrity constraint, Primary key, Candidate key, Alternate key, Foreign key or referential key.

Table, Column, and Row

Data can be stored in a relational database in only one format: in tables. The official name for a table is actually relation, and the term relational model stems from this name. We have chosen to use the term table because SQL uses that word.



PLAYERNO	NAME	INITIALS	TOWN
6	Parmenter	R	Stratford
44	Baker	E	Inglewood
83	Hope	PK	Stratford
100	Parmenter	P	Stratford
27	Collins	DD	Eltham

- **Constraints**

The contents of a table must satisfy certain rules, the so-called integrity constraints (integrity rules). Two examples of integrity constraints are that the player number of a player may not be negative, and two different players may not have the same player number. Integrity constraints can be compared to road signs. They also indicate what is allowed and what is not allowed.

- **Primary Key**

The primary key of a table is a column (or a combination of columns) used as a unique identification of rows in that table. In other words, two different rows in a table may never have the same value in their primary key, and for every row in the table, the primary key must always have one value. The PLAYERNO column in the PLAYERS table is the primary key for this table.

- **Foreign Key**

A foreign key is a column (or combination of columns) in a table in which the population is a subset of the population of the primary key of a table (this does not have to be another table). Foreign keys are sometimes called referential keys.

- **What is SQL?**

As already stated, SQL (Structured Query Language) is a *relational database language*. Among other things, the language consists of statements to insert, update, delete, query, and protect data. The following statements can be formulated with SQL:

■ Insert the address of a new employee.

■ Delete all the stock data for product ABC.

■ Show the address of employee Johnson.

■ Show the sales figures of shoes for every region and for every month.

■ Show how many products have been sold in London the last three months.

■ Make sure that Mr. Johnson cannot see the salary data any longer.

- **Basic Data Types (Text)**

Data type	**Description**
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type Before MySQL version 5.0.3 Varchar datatype can store 255 character, but from 5.0.3 it can be store 65,535 characters. BUT it has a limitation of maximum row size of 65,535 bytes. It means including all columns it must not be more than 65,535 bytes.
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters

- **Basic Data Types (Number)**

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string, allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

- **Basic Data Types (Date)**

Data type	Description
DATE()	A date. Format: YYYY-MM-DD Note: The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'

TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MI:SS Note: The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

Lab No.	Lab Tittle
02	**Managing MySQL databases and tables in MySQL**

Logging on to the MySQL Database Server

- **Login DB from Command Prompt**

```
cd\
cd xampp
cd mysql
cd bin
mysql -u root -p -h 127.0.0.1
```

- **Show Databases and Tables**

```
show databses;
show tables;
```

- **Drop database and table**

```
drop database [database_name];
drop table [table_name];
drop table employees;
```

- **Create database**

```
create database db_lab;
```

- **Use database for selecting database**

```
use db_lab;
```

- **Create a table and Automatic increment values**

```
CREATE TABLE student
(
ID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

- **Describe table student**

```
Describe student;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| ID        | int(11)      | YES  |     | NULL    |       |
| LastName  | varchar(255) | YES  |     | NULL    |       |
| FirstName | varchar(255) | YES  |     | NULL    |       |
| Address   | varchar(255) | YES  |     | NULL    |       |
| City      | varchar(255) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
```

- **Insert value into student table**

```
INSERT INTO student(ID,LastName,FirstName,Address,City)
VALUES(101,'Mahmud','Sakib','Sylhet','Dhaka');

INSERT INTO student(ID,LastName,FirstName,Address,City)
VALUES(102,'Sharmin','Zeseya','Sylhet','Mirpur');
```

- **Find the all value from table student**

```
SELECT * FROM student;
+------+----------+-----------+---------+--------+
| ID   | LastName | FirstName | Address | City   |
+------+----------+-----------+---------+--------+
|  101 | Mahmud   | Sakib     | Sylhet  | Dhaka  |
|  102 | Sharmin  | Zeseya    | Sylhet  | Mirpur |
+------+----------+-----------+---------+--------+
```

Lab No.	Lab Tittle
03	**Implementation of Integrity constraints in MySQL**

Implementation of Primary Key

- **Declaration of PRIMARY KEY**

```
CREATE TABLE PLAYERS(
player_no INT primary key,
player_name varchar(255),
league_no char(4)
);
describe PLAYERS;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| player_no   | int(11)      | NO   | PRI | NULL    |       |
| player_name | varchar(255) | YES  |     | NULL    |       |
| league_no   | char(4)      | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

- **Alternate Declaration of PRIMARY KEY**

```
CREATE TABLE PLAYERS(
player_no INTEGER,
player_name varchar(255),
league_no char(4),
PRIMARY KEY (player_no)
);
describe PLAYERS;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| player_no   | int(11)      | NO   | PRI | NULL    |       |
| player_name | varchar(255) | YES  |     | NULL    |       |
| league_no   | char(4)      | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

- **CREATE Primary Key for TEXT data type**

```
CREATE TABLE PLAYERS(
player_no text(10),
player_name varchar(255),
league_no char(4),
PRIMARY KEY (player_no(10))
);
```

```
describe PLAYERS;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| player_no   | tinytext     | NO   | PRI | NULL    |       |
| player_name | varchar(255) | YES  |     | NULL    |       |
| league_no   | char(4)      | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

- CREATE Composite Key

```
CREATE TABLE diplomas(
student_name text(20) NOT NULL,
course INTEGER NOT NULL,
d_date DATE NOT NULL,
successfull CHAR(1),
location VARCHAR(50),
PRIMARY KEY (student_name(20),course,d_date)
);
```

```
describe diplomas;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| student_name | tinytext    | NO   | PRI | NULL    |       |
| course       | int(11)     | NO   | PRI | NULL    |       |
| d_date       | date        | NO   | PRI | NULL    |       |
| successfull  | char(1)     | YES  |     | NULL    |       |
| location     | varchar(50) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
```

Implementation of UNIQUE

- Unique Constraints

```
CREATE TABLE teams(
team_no INTEGER NOT NULL,
player_no INTEGER NOT NULL,
division CHAR(15) NOT NULL,
PRIMARY KEY (team_no),
UNIQUE (player_no)
);
describe teams;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| team_no   | int(11)  | NO   | PRI | NULL    |       |
| player_no | int(11)  | NO   | UNI | NULL    |       |
| division  | char(15) | NO   |     | NULL    |       |
+-----------+----------+------+-----+---------+-------+
```

- Insert values into table teams

```
INSERT INTO teams(team_no,player_no,division)
VALUES(101,77,'Dhaka'),
      (102,78,'Dhaka'),
      (103, null,'Dhaka');
```

- Find all records from table teams

```
SELECT * FROM teams;
+---------+-----------+----------+
| team_no | player_no | division |
+---------+-----------+----------+
|     101 |        77 | Dhaka    |
|     102 |        78 | Dhaka    |
|     103 |         0 | Dhaka    |
+---------+-----------+----------+
```

- Insert values into table teams

```
INSERT INTO teams(team_no,player_no,division)
VALUES(null,77,'Dhaka');
```

- Find the error message because of primary key (team_no)

```
ERROR 1048 (23000): Column 'team_no' cannot be null
```

- Composite Unique key declaration

```
CREATE TABLE teams(
team_no INTEGER NOT NULL,
player_no INTEGER NOT NULL,
division CHAR(15) NOT NULL,
PRIMARY KEY (team_no),
UNIQUE (player_no,division)
);
```

- Find the structure of table teams

```
describe teams;
+-----------+----------+------+-----+---------+-------+
| Field     | Type     | Null | Key | Default | Extra |
+-----------+----------+------+-----+---------+-------+
| team_no   | int(11)  | NO   | PRI | NULL    |       |
| player_no | int(11)  | NO   | MUL | NULL    |       |
| division  | char(15) | NO   |     | NULL    |       |
+-----------+----------+------+-----+---------+-------+
```

```
INSERT INTO teams(team_no,player_no,division)
VALUES(101,77,'Dhaka'),
      (102,77,'Barisal');
```

```
SELECT * FROM teams;
+---------+-----------+----------+
| team_no | player_no | division |
+---------+-----------+----------+
|     102 |        77 | Barisal  |
|     101 |        77 | Dhaka    |
+---------+-----------+----------+
```

Implementation of UNIQUE

- **Firstly, create a table**

```
CREATE TABLE players(
player_no INTEGER NOT NULL,
name CHAR(15) NOT NULL,
initials CHAR(3) NOT NULL,
birth_date DATE,
sex CHAR(1) NOT NULL,
joined SMALLINT NOT NULL,
street VARCHAR(30) NOT NULL,
PRIMARY KEY(player_no)
);
```

- **Declaration of foreign key constraints**

```
CREATE TABLE teams(
team_no INTEGER NOT NULL,
player_no INTEGER NOT NULL,
division CHAR(6) NOT NULL,
PRIMARY KEY (team_no),
FOREIGN KEY (player_no)
REFERENCES PLAYERS(player_no)
);
```

- **Find the structure of table teams**

```
describe teams;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| team_no   | int(11) | NO   | PRI | NULL    |       |
| player_no | int(11) | NO   | MUL | NULL    |       |
| division  | char(6) | NO   |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
```

Lab No.	Lab Tittle
04	**Modifying MySQL databases and Updating Data in MySQL Table**

Table modification using alter table

- **Create a table and Automatic increment values**

```
CREATE TABLE Persons
(
ID int NOT NULL AUTO_INCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (ID)
);
```

- **Mysql Add Column Examples**

```
ALTER TABLE Persons
ADD COLUMN email varchar(200);
```

- **DROP an attributes/column from table persons**

```
ALTER TABLE Persons DROP COLUMN email;
```

- **Add an attributes/column to table persons in any position of column**

```
ALTER TABLE Persons
ADD COLUMN email varchar(200) AFTER FirstName;
```

- **Add an attributes/column to table persons in the first column**

```
ALTER TABLE Persons
ADD COLUMN Serial_num varchar(200) FIRST;
```

- **Add multiple attributes/column to table persons in single command**

```
ALTER TABLE Persons
ADD COLUMN Salary varchar(200) NOT NULL,
ADD COLUMN entry_date date NOT NULL;
```

- **DROP multiple attributes/column from table persons**

```
ALTER TABLE Persons
DROP COLUMN Address,
DROP COLUMN email;
```

- **Changing columns constraints using MySQL ALTER TABLE statement**

```
ALTER TABLE Persons_info
CHANGE COLUMN salary salary INT NOT NULL;
```

- **Changing columns name using MySQL ALTER TABLE statement**

```
ALTER TABLE Persons_info
CHANGE COLUMN salary per_salary INT NOT NULL;
```

Inserting data into tables using MySQL INSERT statement

- Create a table person_info table

```
CREATE TABLE person_info(
ID int primary key,
LastName varchar(255)NOT NULL,
FirstName varchar(255),
per_salary int NOT NULL,
entry_date date NOT NULL
);
```

- Insert values into person_info table

```
INSERT INTO Persons_info(ID, LastName,FirstName,per_salary,entry_date)
VALUES (1235678943212,'Cook','Alex','560000','2017-02-15');

INSERT INTO Persons_info(ID, LastName,FirstName,per_salary,entry_date)
VALUES (120987,'Shuvo','Anudhuti','860000','2017-02-14');
```

- Find all records from person_info

```
SELECT * FROM Persons_info;
+------------+----------+-----------+------------+------------+
| ID         | LastName | FirstName | per_salary | entry_date |
+------------+----------+-----------+------------+------------+
|     120987 | Shuvo    | Anudhuti  |     860000 | 2017-02-14 |
| 2147483647 | Cook     | Alex      |     560000 | 2017-02-15 |
+------------+----------+-----------+------------+------------+
```

- MySQL copy table examples

```
CREATE TABLE IF NOT EXISTS person_info_backup
SELECT * FROM Persons_info;
```

- Find all records from new copied table

```
SELECT * FROM person_info_backup;
+------------+----------+-----------+------------+------------+
| ID         | LastName | FirstName | per_salary | entry_date |
+------------+----------+-----------+------------+------------+
|     120987 | Shuvo    | Anudhuti  |     860000 | 2017-02-14 |
| 2147483647 | Cook     | Alex      |     560000 | 2017-02-15 |
+------------+----------+-----------+------------+------------+
```

Updating data using MySQL UPDATE statement

- UPDATE a column single value

```
UPDATE employees
SET salary = 30000
WHERE emp_no = 1015312009;
```

- UPDATE a multiple columns single value

```
UPDATE employees
SET first_name = 'Taskin',
    last_name = 'Ahmed'
WHERE emp_no = 1015312009;
```

- Delete a record from a table

```
DELETE FROM employees
WHERE emp_no = 1015312008;
```

Removing data using MySQL DELETE statement

- **Delete all records from a table**

```
DELETE FROM employees;

SELECT * FROM employees;
Empty set (0.00 sec)
```

Lab No.	Lab Tittle
05	**Querying and Filtering data in MySQL Table**

Using MySQL SELECT statement to query data

- **Create a table employees**

```
CREATE TABLE employees (
    emp_no      INT               NOT NULL,
    birth_date  DATE              NOT NULL,
    first_name  VARCHAR(14)       NOT NULL,
    last_name   VARCHAR(16)       NOT NULL,
    gender      ENUM ('M','F')    NOT NULL,
    salary      INT               NOT NULL,
    entry_date  datetime          NOT NULL    DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (emp_no)
);
```

- **Insert Multiple VALUES at a time**

```
INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender, salary)
VALUES(1015312001, '1989-08-28', 'Rina','Khanam','F', 45000),
      (1015312002, '1988-07-19', 'Sakib','Hasan','M', 67000),
      (1015312003, '1991-05-23', 'Sabbir','Rahman','M', 32000);
```

- **Insert Single Values {Must have same values as attributes number}**

```
INSERT INTO employees VALUES (1015312008, '1991-05-23', 'Sabbir','Rahman','M',24000, '2017-11-11');
INSERT INTO employees VALUES (1015312009, '1991-05-23', 'Sabbir','Rahman','M',25600, '2017-11-11 21:44:35');
```

- **View data from table**

```
SELECT * FROM employees;
+------------+------------+------------+-----------+--------+--------+---------------------+
| emp_no     | birth_date | first_name | last_name | gender | salary | entry_date          |
+------------+------------+------------+-----------+--------+--------+---------------------+
| 1015312001 | 1989-08-28 | Rina       | Khanam    | F      |  45000 | 2017-02-14 22:29:31 |
| 1015312002 | 1988-07-19 | Sakib      | Hasan     | M      |  67000 | 2017-02-14 22:29:31 |
| 1015312003 | 1991-05-23 | Sabbir     | Rahman    | M      |  32000 | 2017-02-14 22:29:31 |
| 1015312004 | 1990-02-14 | mushfiqur  | Rahman    | M      |  42000 | 2017-02-14 22:29:22 |
| 1015312008 | 1991-05-23 | Sabbir     | Rahman    | M      |  24000 | 2017-11-11 00:00:00 |
| 1015312009 | 1991-05-23 | Sabbir     | Rahman    | M      |  25600 | 2017-11-11 21:44:35 |
+------------+------------+------------+-----------+--------+--------+---------------------+
```

Eliminating duplicate rows with DISTINCT Operator

- **Using MySQL DISTINCT to Eliminate Duplicates**

```
SELECT DISTINCT first_name, last_name FROM employees;
```

Filtering rows using MySQL WHERE

- **MySQL WHERE for INETEGER type value**

```
SELECT emp_no, first_name, last_name,salary, entry_date
FROM employees
WHERE emp_no = 1015312003;
```

- **MySQL WHERE for String type value**

```
SELECT emp_no, first_name, last_name,salary, entry_date
FROM employees
WHERE first_name = 'Sakib';
```

Using comparison operators (<,>, <=,>=, <>)

- **Example-1**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE salary >=45000;
```

- **Example-2**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE salary <> 45000;
```

Lab No.	Lab Tittle
06	**Querying and Filtering data in MySQL Table (Extended)**

Using logical operators (AND, OR, NOT)

- **Create a table employees**

```
CREATE TABLE employees (
    emp_no      INT              NOT NULL,
    birth_date  DATE             NOT NULL,
    first_name  VARCHAR(14)      NOT NULL,
    last_name   VARCHAR(16)      NOT NULL,
    gender      ENUM ('M','F')   NOT NULL,
    salary      INT              NOT NULL,
    entry_date  datetime         NOT NULL    DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (emp_no)
);
```

- **Insert Multiple VALUES at a time**

```
INSERT INTO employees (emp_no, birth_date, first_name, last_name, gender, salary)
VALUES(1015312001, '1989-08-28', 'Rina','Khanam','F', 45000),
     (1015312002, '1988-07-19', 'Sakib','Hasan','M', 67000),
     (1015312003, '1991-05-23', 'Sabbir','Rahman','M', 32000);
```

- **Insert Single Values {Must have same values as attributes number}**

```
INSERT INTO employees VALUES (1015312008, '1991-05-23', 'Sabbir','Rahman','M',24000, '2017-11-11');
INSERT INTO employees VALUES (1015312009, '1991-05-23', 'Sabbir','Rahman','M',25600, '2017-11-11 21:44:35');
```

- **MySQL AND operator examples**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name = 'Rina' AND last_name = 'Khanam';
```

- **MySQL OR operator examples**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name = 'Rina' OR last_name = 'Khan';
```

- **Operator precedence MySQL evaluates the OR operators after the AND operators**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name = 'Rina' OR last_name = 'Rahman' AND salary <=40000;
```

- **To change the order of evaluation, you use the parentheses, for example:**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE (first_name = 'Rina' OR last_name = 'Rahman') AND salary <=40000;
```

- **MySQL creates result for OR**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name = 'Rina' OR last_name = 'Rahman';
```

Using limit (ORDER BY, ASC, DESC)

- **Select the first 3 customers**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
LIMIT 3;
```

- **Select all attributes**

```
SELECT emp_no, first_name, last_name, salary
FROM employees;
```

- **Find 4 records without first 2 records**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
LIMIT 2,4;
```

- **Using MySQL LIMIT to get the highest 3 values**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY salary DESC
LIMIT 3;
```

- **Using MySQL ORDER BY and DESC to get the Descending Order salary**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY salary DESC;
```

- **Using MySQL LIMIT to get the highest values**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY salary DESC
LIMIT 1;
```

- **Using MySQL LIMIT to get the highest 2nd values**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY salary DESC
LIMIT 1,1;
```

- **Using MySQL LIMIT to get the lowest 3 values**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY salary ASC
LIMIT 3;
```

- **Using MySQL ORDER BY and ASC to get the Ascending Order salary**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY salary ASC;
```

Between, Not Between In, Not In

- **MySQL IN examples Like OR operator**

```
SELECT emp_no, first_name, last_name, salary
FROM employees
WHERE salary IN (32000,45000);
```

- MySQL NOT IN examples
```
SELECT emp_no, first_name, last_name, salary
FROM employees
WHERE salary NOT IN (32000,45000,25600);
```
- MySQL BETWEEN examples
```
SELECT emp_no, first_name, last_name, salary
FROM employees
WHERE salary BETWEEN 20000 AND 43000;
```
- MySQL BETWEEN to get exact values
```
SELECT emp_no, first_name, last_name, salary
FROM employees
WHERE salary BETWEEN 25600 AND 42000;
```
- MySQL NOT BETWEEN to get exact values
```
SELECT emp_no, first_name, last_name, salary
FROM employees
WHERE salary NOT BETWEEN 25600 AND 42000;
```
- MySQL BETWEEN with dates example
```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE entry_date
BETWEEN CAST('2017-10-01' AS DATE) AND CAST('2017-12-01' AS DATE);
```
- View date with different format
```
SELECT emp_no, first_name, last_name, salary, entry_date,
DATE_FORMAT(entry_date, '%M %D, %Y') AS new_style
FROM employees;
```

Using MySQL LIKE operator to select data based on patterns

- ✓ **MySQL LIKE examples**
- ✓ **The percentage (%) wildcard allows you to match any string of zero or more characters.**
- ✓ **The underscore (_) wildcard allows you to match any single character.**

- Find employees name who has first name starting with 'm'
```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name LIKE 'm%';
```
- Find employees name who has first name ending with 'r'
```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name LIKE '%r';
```
- Find employees name who has first name contains 'bb'
```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name LIKE '%bb%';
```

- **Find employees name who has first name contains first letter ‘r’ and fourth letter ‘a’**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name LIKE 'r__a';
```

- **Example-2**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name LIKE '%a_i%';
```

- **MySQL LIKE operator with NOT operator**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name NOT LIKE 'r__a';
```

- **Example-2**

```
SELECT emp_no, first_name, last_name, salary, entry_date
FROM employees
WHERE first_name NOT LIKE '%r';
```

Checking NULL values

- **NULL value check**

```
SELECT * FROM employees
WHERE gender IS NULL;
```

Lab No.	Lab Tittle
07	**Implementation of MySQL Aggregate Function**

Using AVG(), SUM(), MIN(), MAX(), COUNT()

- **Create a tbake product _order_info**

```
CREATE TABLE product_order_info(
    product_no INT                NOT NULL AUTO_INCREMENT,
    product_name VARCHAR(255)     NOT NULL,
    product_type ENUM('electronics','stationary','food','bevarage'),
    product_price FLOAT(10,2)     NOT NULL,
    product_quantity SMALLINT     NOT NULL,
    order_date DATETIME           NOT NULL    DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (product_no)
);
```

- **INSERT values into table**

```
INSERT INTO product_order_info(product_no, product_name, product_type, product_price, product_quantity)
VALUES (1001,'Laptop','electronics',67000,'1');
```

```
INSERT INTO product_order_info(product_name, product_type, product_price, product_quantity)
VALUES ('Laptop','electronics',67000,'1'),
       ('Mobile','electronics',23500,'1'),
       ('Watch','electronics',8650,'2'),
       ('Butter','stationary',50,'5'),
       ('Cocacola','bevarage',35,'2'),
       ('SevenUp','bevarage',55,'1');
```

- **--AVG function**
- **--The AVG function calculates the average value of a set of values.**
- **--It ignores NULL values in the calculation.**

```
SELECT AVG(product_price) avg_product_price
FROM product_order_info;
--OR
SELECT AVG(product_price) AS avg_product_price
FROM product_order_info;
```

- **COUNT function returns the number of the rows in a table.**

```
SELECT COUNT(product_no) AS total_order
FROM product_order_info;
```

- **COUNT function returns the number of the rows in a table.**

```
SELECT COUNT(*)
FROM product_order_info
WHERE product_type = 'electronics';
```

- **COUNT function returns the number of the rows of specific items.**

```
SELECT COUNT(*),product_type, product_name, product_price
FROM product_order_info
GROUP BY product_type;
```

- **Example-2**

```
SELECT COUNT(*),product_type, product_name, product_price
FROM product_order_info
GROUP BY product_name;
```

- Example-3

```
SELECT COUNT(product_quantity),product_type, product_name, product_price
FROM product_order_info
GROUP BY product_name;
```

- **--The SUM function returns the sum of a set of values.**
- **--The SUM function ignores NULL values.**
- **--If no matching row found, the SUM function returns a NULL value.**

```
SELECT SUM(product_price) AS total_price
FROM product_order_info;
```

- **To get the total sales of each product,**

```
SELECT product_no, product_name,product_price,product_quantity,
SUM(product_price * product_quantity) AS total_per_product
FROM product_order_info
GROUP BY product_no;
```

- **Example-2**

```
SELECT product_no, product_name,product_price,product_quantity,
SUM(product_price * product_quantity) AS total_per_product
FROM product_order_info;
```

- **Example-3**

```
SELECT product_no, product_name,product_price,product_quantity,
SUM(product_price * product_quantity) AS total_per_product
FROM product_order_info
GROUP BY product_name;
```

- **MAX function returns the maximum value in a set of values.**

```
SELECT MAX(product_price) max_price
FROM product_order_info;
```

- **MIN function returns the minimum value in a set of values.**

```
SELECT MIN(product_price) min_price
FROM product_order_info;
```

Using LENGTH(), UCASE/ UPPER CASE(), LCASE/ LOWER CASE(), MID(), ROUND/ FLOOR/ CELLING(), CONCAT()

- **MySQL LENGTH function**

```
SELECT product_no, product_name,product_price,LENGTH(product_price)
FROM product_order_info;
```

- **Example-2**

```
SELECT product_no, product_name,product_price
FROM product_order_info
WHERE LENGTH(product_price)>5;
```

- **UCASE function**

```
SELECT product_no, product_name, product_type, UCASE(product_name)
FROM product_order_info;
```

- **LCASE function**

```
SELECT product_no, product_name, product_type, LCASE(product_name)
FROM product_order_info;
```

- FLOOR function

```
SELECT product_no, product_name, product_type, FLOOR(product_price)
FROM product_order_info;
```

- CELLING function

```
SELECT product_no, product_name, product_type, CEIL(product_price)
FROM product_order_info;
```

- ROUND function

```
SELECT product_no, product_name, product_type, ROUND(product_price)
FROM product_order_info;
```

- MID function

```
SELECT product_no, product_name, product_type, MID(product_price,1,3)
FROM product_order_info;
```

- Example-2

```
SELECT product_no, product_name, product_type, MID(product_price,2,4)
FROM product_order_info;
```

- CONCAT function

```
SELECT product_no, product_name, product_type, CONCAT(product_name,' ', product_type)
FROM product_order_info;
```

Sorting data using ORDER BY, GROUP BY

- MySQL ORDER BY examples

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY first_name;
```

- MySQL ORDER BY is by default Ascending order

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY salary;
```

- MySQL ORDER BY to find descending order

```
SELECT emp_no, first_name, last_name, salary
FROM employees
ORDER BY salary desc;
```

Lab No.	Lab Tittle
08	**Implementation of Relational Databases (Join Function)**

Extracting Information from Multiple Table (Union, Union All)

- **Create a table student**

```
CREATE TABLE student
(
    S_ID int                                    NOT NULL,
    FirstName varchar(255)                      NOT NULL,
    LastName varchar(255)                       NOT NULL,
    Address varchar(255)                        NOT NULL,
    Department ENUM('CSE','EEE','TEX')          NOT NULL,
    AdmissionDate datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (S_ID)
);
```

- **Insert values into student table**

```
INSERT INTO student(S_ID, FirstName,LastName,Address,Department)
VALUES(142002015, 'Zeseya', 'Sharmin', 'Dhaka','CSE'),
      (142002001, 'Sakib', 'Hasan', 'Natore','CSE'),
      (162002002, 'Asef', 'Tajwar', 'Rangpur','EEE'),
      (162002003, 'Maruf', 'Hasan', 'Barisal','EEE'),
      (172002002, 'Ashek', 'Farabi', 'Gazipur','TEX'),
      (173002003, 'Ismile', 'Hasan', 'Barisal','TEX');
```

- **Create another table department**

```
CREATE TABLE department (
    dept_id       INT                          NOT NULL AUTO_INCREMENT,
    dept_name enum('CSE','EEE','TEX')          NOT NULL,
    dept_location VARCHAR(255)                 NOT NULL,
    PRIMARY KEY (dept_id)
);
```

- **Insert values into department table**

```
INSERT INTO department(dept_id, dept_name,dept_location)
VALUES(101, 'CSE', 'Building-2'),
      (102, 'EEE', 'Building-2'),
      (103, 'TEX', 'Building-1');
```

- **Create another table course_registrstion**

```
CREATE TABLE course_resgistration(
    reg_serial INT                     NOT NULL AUTO_INCREMENT,
    course_code VARCHAR(255)           NOT NULL,
    Course_title VARCHAR(255)          NOT NULL,
    dept_id INT                        NOT NULL,
    s_id INT                           NOT NULL,
    PRIMARY KEY (reg_serial)
);
```

- Insert values into course_registration table

```
INSERT INTO course_resgistration(course_code,Course_title,dept_id,s_id)
VALUES('CSE 311', 'Computer Networks', 101,142002015),
      ('CSE 311', 'Computer Networks', 101,142002001),
      ('EEE 301', 'Electrical Circuit', 201,162002002),
      ('TEX 201', 'Aparales', 301,172002002),
      ('CSE 312', 'Computer Networks Lab', 101,142002015),
      ('CSE 207', 'Algorithm', 101,142002001);
```

- /*join_table:
- table_reference [INNER | CROSS] JOIN table_factor [join_condition]
- | table_reference STRAIGHT_JOIN table_factor
- | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
- | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
- | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor */

```
SELECT S_ID
FROM student
UNION
SELECT s_id
FROM course_resgistration;
```

- --Example of UNION ALL
- --Duplicate data exist

```
SELECT S_ID
FROM student
UNION ALL
SELECT s_id
FROM course_resgistration;
```

Join, Inner Join, Left Join, Right Join, Where, Group by

- INNER JOIN example

```
SELECT student.S_ID, student.FirstName, student.Department
FROM student
INNER JOIN course_resgistration ON student.S_ID=course_resgistration.s_id;
```

- Example-2

```
SELECT student.S_ID, student.FirstName, student.Department
FROM student
INNER JOIN department ON student.department = department.dept_name;
```

- INNER JOIN with WHERE clause

```
SELECT student.S_ID, student.FirstName, student.Department
FROM student
INNER JOIN course_resgistration ON student.S_ID = course_resgistration.s_id
WHERE course_resgistration.s_id = 142002015;
```

- Multiple Inner Join

```
SELECT student.S_ID, student.FirstName, student.Department,department.dept_id
FROM student
INNER JOIN department ON student.department = department.dept_name;
```

- Example-2

```
SELECT department.dept_id, dept_name, course_resgistration.course_code
FROM department
INNER JOIN course_resgistration ON department.dept_id= course_resgistration.dept_id;
```

- **Multiple Inner Join**

```
SELECT student.S_ID, student.FirstName, student.Department,
department.dept_id,course_resgistration.course_code
FROM student
INNER JOIN department ON student.department = department.dept_name
INNER JOIN course_resgistration ON department.dept_id= course_resgistration.dept_id;
```

- **INNER JOIN using GROUP BY for eleminiting duplicate records.**

```
SELECT student.S_ID, student.FirstName, student.Department,
department.dept_id,course_resgistration.course_code
FROM student
INNER JOIN department ON student.department = department.dept_name
INNER JOIN course_resgistration ON department.dept_id= course_resgistration.dept_id
GROUP BY S_ID;
```

- **INNER JOIN using ORDER BY for ascending orders of records**

```
SELECT student.S_ID, student.FirstName, student.Department,department.dept_id
FROM student
INNER JOIN department ON student.department = department.dept_name
INNER JOIN course_resgistration ON department.dept_id= course_resgistration.dept_id
ORDER BY FirstName;
```

- **LEFT JOIN example**

```
SELECT student.S_ID, student.FirstName, student.Department
FROM student
LEFT JOIN course_resgistration
ON student.S_ID=course_resgistration.s_id;
```

Lab No.	Lab Tittle
09	**Implementation of Databases Triggers**

Introduction to triggers

- **Create the CHANGES table.**

```
CREATE TABLE CHANGES
      (USER              CHAR(30) NOT NULL,
       CHA_TIME          TIMESTAMP NOT NULL,
       CHA_PLAYERNO      SMALLINT NOT NULL,
       CHA_TYPE          CHAR(1) NOT NULL,
       CHA_PLAYERNO_NEW  INTEGER,
       PRIMARY KEY       (USER, CHA_TIME,
                         CHA_PLAYERNO, CHA_TYPE))
```

- **Create the trigger that updates the CHANGES table automatically as new rows are added to the PLAYERS table.**

```
CREATE TRIGGER INSERT_PLAYERS
   AFTER
   INSERT ON PLAYERS FOR EACH ROW
   BEGIN
      INSERT INTO CHANGES
         (USER, CHA_TIME, CHA_PLAYERNO,
          CHA_TYPE, CHA_PLAYERNO_NEW)
      VALUES (USER, CURDATE(), NEW.PLAYERNO, 'I', NULL);
   END
```

Lab No.	Lab Tittle
10	**Implementation of creating users and data Security**

- User creating, user removing
- Changing of user name, passwords
- Granting table and column privileges
- Granting databases privileges, user privileges,
- Restricting privileges, recording privileges, revoking privileges
- Security through view

o **Adding and Removing Users**

```
CREATE USER
   'CHRIS'@'localhost' IDENTIFIED BY 'CHRISSEC',
   'PAUL'@'localhost' IDENTIFIED BY 'LUAP'
```

o **Three new users and then show the contents of the USERS catalog view.**

```
CREATE USER
   'CHRIS1'@'sql.r20.com' IDENTIFIED BY 'CHRISSEC1',
   'CHRIS2'@'%' IDENTIFIED BY 'CHRISSEC2',
   'CHRIS3'@'%.r20.com' IDENTIFIED BY 'CHRISSEC3'

SELECT   *
FROM     USERS
WHERE    USER_NAME LIKE '''CHRIS%'
ORDER    BY 1
```

o **Drop the user JIM.**

```
DROP USER JIM
```

o **Change the names of the users CHRIS1 and CHRIS2 to COMBO1 and COMBO2, respectively, and then show the contents of the USERS catalog view.**

```
RENAME USER
   'CHRIS1'@'sql.r20.com' TO 'COMBO1'@'sql.r20.com',
   'CHRIS2'@'%' TO 'COMBO2'@'sql.r20.com'
```

o **See All**

```
SELECT   *
FROM     USERS
WHERE    USER_NAME LIKE '''COMBO%'
ORDER    BY 1
```

o **Changing Passwords**

```
SET PASSWORD FOR 'JOHN'= PASSWORD('JOHN1')
```

o **Change the password of ROB to ROBSEC.**

```
SET PASSWORD FOR ROB = PASSWORD('ROBSEC')
```

o **Give JAMIE the SELECT privilege on the PLAYERS table.**

```
GRANT   SELECT
ON      PLAYERS
TO      JAMIE
```

- **Give the new user BOB the SELECT privilege on the PLAYERS table.**

```
GRANT    SELECT
ON       PLAYERS
TO       'BOB'@'localhost' IDENTIFIED BY 'BOBPASS'
```

- **Give JAMIE and PETE the INSERT and UPDATE privileges for all columns of the TEAMS table.**

```
GRANT    INSERT, UPDATE
ON       TEAMS
TO       JAMIE, PETE
```

- **Give PETE the SELECT privilege for all tables in the TENNIS database.**

```
GRANT    SELECT
ON       TENNIS.*
TO       PETE
```

- **Give JIM the privilege to create, update, and remove new tables and views in the TENNIS database.**

```
GRANT    CREATE, ALTER, DROP, CREATE VIEW
ON       TENNIS.*
TO       JIM
```

- **Give ALYSSA the SELECT and INSERT privileges for all tables in the current database.**

```
GRANT    SELECT, INSERT
ON       *
TO       ALYSSA
```

Granting User Privileges

- **Give MAX the CREATE, ALTER, and DROP privileges for all tables of all databases.**

```
GRANT    CREATE, ALTER, DROP
ON       *.*
TO       MAX
```

- **Give ALYSSA the privilege to create new users.**

```
GRANT    CREATE USER
ON       *.*
TO       ALYSSA
```

Lab No.	Lab Tittle
11	**Implementation of Databases Transactions and Multiuser Usage**

So far in this book, we have assumed that you are the only user of the database. If you do the examples and exercises at home, that assumption is probably correct. But if you work with MySQL in your company, for example, the odds are good that you share the database with many other users. We call this multiuser usage as opposed to single-user usage. In a multiuser environment, you should not be aware that other users are accessing the database concurrently because MySQL hides this from you as much as possible. The following question might arise: What happens if I access a row that is already in use by someone else? This chapter answers that question. We start with a concept that forms the basis of multiuser usage: the transaction (also called unit of work). We also discuss the concepts savepoint, lock, deadlock, and isolation level, and we consider the LOCK TABLE statement.

Not all storage engines support transactions; for example, InnoDB and BDB do, but MyISAM and MEMORY do not. Therefore, this chapter assumes that you created the tables with one of the storage engines that does support transactions.

This chapter looks inside MySQL. If that does not interest you, you can skip this chapter. For those who will develop real-life applications with MySQL, we recommend studying this chapter carefully.

To illustrate these concepts, see the following series of statements that are entered consecutively. It is not important whether these statements are entered interactively (with MySQL, for example) or are embedded within a host

language program:

1. INSERT ...
2. DELETE ...
3. ROLLBACK WORK
4. UPDATE ...
5. ROLLBACK WORK
6. INSERT ...
7. DELETE ...
8. COMMIT WORK

9. UPDATE ...

10. end of program

Delete all data for player 6. We assume that no foreign keys have been defined.

```
DELETE FROM PLAYERS WHERE PLAYERNO = 6
DELETE FROM PENALTIES WHERE PLAYERNO = 6
DELETE FROM MATCHES WHERE PLAYERNO = 6
DELETE FROM COMMITTEE_MEMBERS WHERE PLAYERNO = 6
UPDATE TEAMS SET PLAYERNO = 83 WHERE PLAYERNO = 6
```

12	**Design and Implementation of a Group Project – Presentation**

- ✓ Students have to submit the lab report before final examination.
- ✓ Students have to appear in the assessment examination.
- ✓ Students have to make a presentation individually.
- ✓ Students have to complete one database application.

13	**Viva and Final Examination**

- ✓ Each student has to appear in the viva voce for 15 Marks
- ✓ Each student has to appear in the final examination for 30 Marks