CS105.3
Database Management Systems

SQL Course Manual

Ms.Manoja Weerasekara

Faculty of Computing

NSBM Green University

Learning SQL

OBJECTIVE:

In this section, we will discuss how to implement your database and handle a few core operations related to it.

You are aware of C as a language used to write programs. Do you think we can use C to communicate with your database?

The answer is "**NO**".

Why? Because the database cannot understand this language. Like you will not understand if someone talks in a different language that you're not familiar with ☺.

So, what is the language we can use to communicate with a database? It's **SQL**

Introduction to SQL: Structured Query Language

IBM developed SQL to interact with databases. SQL is an ANSI (American National Standards Institute) standard language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. SQL works with database programs like; Microsoft SQL Server, MySQL, IBM-DB2, Informix, Microsoft Access, etc. When a user wants to get some information from a database file, he can issue a **query**. A query is a user request to retrieve data or information with a specific condition. The result of the query will then be stored in the form of a table. The program will go through all the records in the database file and select those records that satisfy the condition(searching). SQL is a query language that allows the user to specify the conditions. (instead of algorithms). There are many different versions of the SQL language.

In this book, we'll show you how to use **MySQL**'s flavor of SQL to create databases and store and modify data.

*HINT: WHEN YOU'RE READING THE TEXT LOOK INTO THE **BOLDED** TERMS. CHECK WHETHER YOU UNDERSTAND THEM.*

SQL Database Tables

A **database** most often contains one or more **tables**. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

LastName	FirstName	City
Perera	Onali	Colombo
Silva	Chaminda	Galle
Rathnayake	Rani	Kandy

The table above contains three records (one for each person) and three columns (LastName, FirstName, and City).

SQL Queries

With SQL, we can **query** a database and have a result set returned. A query like this:

```
SELECT LastName FROM Persons;
```

This query will give a **result set** like this:

LastName
Perera
Silva
Rathnayake

SQL Syntax

SQL keywords are NOT case sensitive:

"select" is the same as "SELECT".

Some database systems require a semicolon at the end of each SQL statement.

The semicolon is the standard way to separate each SQL statement in database systems. It allows more than one SQL statement to be executed in the same call to the server.

SQL Command Types

SQL language is divided into three types of primary language statements.

- DML (Data Manipulation Language)

- DDL (Data Definition Language)
- DCL (Data Control Language)

SQL Data Manipulation Language (DML)

SQL (Structured Query Language) is a syntax for executing queries. However, the SQL language also includes a syntax to update, insert, and delete records. These query and update commands together form the Data Manipulation Language (DML) part of SQL:

- SELECT - extracts data from a database table
- UPDATE - updates data in a database table
- DELETE - deletes data from a database table
- INSERT INTO - inserts new data into a database table

SQL Data Definition Language (DDL)

The Data Definition Language (DDL) permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables. The most common DDL statements in SQL are:

- CREATE TABLE - creates a new database table
- ALTER TABLE - alters (changes) a database table
- DROP TABLE - deletes a database table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

SQL Data Control Language (DCL)

DCL is the short name of Data Control Language which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

- GRANT – allow users access privileges to database
- REVOKE – withdraw users access privileges given by using the GRANT command

Time to Learn while practising:

NOTE: Look into the appendix 1 to get to know how to install the required software for these practical sessions.

Basic Operations

Task 01: What are the databases currently available in the server?

Syntax:

```
SHOW databases;
```

Sample output:

Note: In the output, you will see 'mysql'. It is the database which stores users' password, and it is created and used by the system. Let us create your first database.

Task 02: Create a database (make a directory) whose name is MyDB

Syntax:

```
CREATE database <database name>;
```

E.g. CREATE database MyDB;

Task 03: Now we will select the newly created database to use for other operations

Syntax:

```
USE <database name>;
```

E.g. Use MyDB;

Look into the command line it will say "Database changed". What has changed?

Task 04: What tables currently stored in the MyDB database?

Syntax:

```
SHOW TABLES;
```

It will print 'Empty set (0.00 sec)' as we have not created any tables in the database yet.

Task 05: Delete (drop) MyDB database

Syntax:

```
DROP DATABASE <database name>;
```

E.g. Drop database myDB;

Data Types

The data type specifies what type of data the column can hold. The table below contains the most common data types in SQL:

Data Type	Description
integer(size) int(size) smallint(size) tinyint(size)	Hold integers only. The maximum number of digits is specified in parenthesis.
decimal(size,d) numeric(size,d)	Hold numbers with fractions. The maximum number of digits is specified in "size". The maximum number of digits to the right of the decimal is specified in "d".
char(size)	Holds a fixed-length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis.
varchar(size)	Holds a variable-length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis.
date(yyyymmdd)	Holds a date

Note: There are many other data types and formats.

Field Size

Determines the maximum number of characters that can be saved in a particular field as an attribute value. Field size is defined with the data type.

E.g. Let's assume you want to declare a char type variable with size of 4 you will write it as CHAR (4) and if it is varchar with the size of 20 you need to write it as VARCHAR (20).

It is not required to specify the field sizes for INTEGER and DATE data types.

Task 06: Create a Table

To create a table in a database:

Syntax:

```
CREATE TABLE table_name
(
column_name1 data_type(field_size),
column_name2 data_type(field_size),
.....
);
```

E.g. This example demonstrates how you can create a table named "Person", with four columns. The column names will be "LastName", "FirstName", "Address", and "Age":

This example demonstrates how you can specify a maximum length/field size for columns:

```
CREATE TABLE Person
(
LastName varchar(30),
FirstName varchar(30),
Address varchar(100),
Age int(3)
);
```

It is not required to specify the field sizes for INTEGER

The following examples show a table created with adding few other constrains, e.g. Primary key, Not Null, Auto Increment.

```
CREATE TABLE Book(
  book_id INT NOT NULL AUTO_INCREMENT,
  book_title VARCHAR(100) NOT NULL,
  book_author VARCHAR(40) NOT NULL,
  submission_date DATE,
  PRIMARY KEY (book_id )
);
```

Field Attribute **NOT NULL** is being used because we do not want this field to be NULL. So, if a user will try to create a record with a NULL value, then MySQL will raise an error.

Field Attribute **AUTO_INCREMENT** tells MySQL to go ahead and add the next available number to the id field.

Keyword **PRIMARY KEY** is used to define a column as a primary key. You can use multiple columns separated by a comma to define a primary key.

Task 07: Creating relationships among tables: Foreign Key Constraint

Consider the following example below:

"Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

You can see that the Primary Key of Person appears at the Orders table as a **Foreign Key.** How to set up this relationship?

```
CREATE TABLE  Persons (
   PersonID int NOT NULL,
   LastName varchar(15) NOT NULL,
   FirstName varchar(15) NOT NULL,
   Age int,
   PRIMARY KEY (PersonID),
   );

CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Strictly speaking, for a field to be a foreign key, it needs to be defined as such in the database definition. In order to create a foreign key, you need the following:

- Both tables need to be InnoDB tables.

- To use the syntax FOREIGN KEY(fk_fieldname) REFERENCES table_name (fieldname)
- The field being declared a foreign key needs to be declared as an index in the table definition

Task 08: Delete and Truncate a database table

Delete a Table

This will delete the entire table (the table structure, attributes, and indexes will also be deleted).

Syntax:

```
DROP TABLE <Table_name>;
```

E.g. Drop table book;

Truncate a Table

What if we only want to get rid of the data inside a table, and not the table itself? Use the TRUNCATE TABLE command (deletes only the data inside the table):

```
TRUNCATE TABLE <table_name>;
```

Now, can you try to delete the 'Persons' table? What is the result?

More DML type Operations:

Task 09: The INSERT INTO Statement

The INSERT INTO statement is used to insert new rows into a table.

Syntax

```
INSERT INTO table_name
VALUES (value1, value2,....);
```

You can also specify the columns for which you want to insert data:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....);
```

Insert a New Row

This "Persons" table:

LastName	FirstName	City
Perera	Onali	Colombo

And this SQL statement:

```
INSERT INTO Persons
VALUES ('Herath', 'Chaminda', ‘Kandy’);
```

Will give this result:

LastName	FirstName	City
Perera	Onali	Colombo
Herath	Chaminda	Kandy

Task 10: Insert Data in Specified Columns

This "Persons" table:

LastName	FirstName	City
Perera	Onali	Colombo
Herath	Chaminda	Kandy

and this SQL statement:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Weerasekara', ‘Galle’);
```

will give this result:

LastName	FirstName	City
Perera	Onali	Colombo
Herath	Chaminda	Kandy
Weerasekara		Galle

Task 11: Select Specified Columns

Syntax:

```
SELECT column1, column2,...
FROM  table_name;
```

To select the content of columns named "LastName" and "FirstName", from the database table called "Persons", use a SELECT statement like this:

```
SELECT LastName,FirstName FROM Persons;
```

The database table "Persons":

LastName	FirstName	City
Perera	Onali	Colombo
Herath	Chaminda	Kandy
Weerasekara		Galle

The result

LastName	FirstName
Perera	Onali
Herath	Chaminda
Weerasekara	

Task 12: Select All Columns

Syntax:

```
SELECT * FROM table_name;
```

To select all columns from the "Persons" table, use a * symbol instead of column names, like this:

```
SELECT * FROM Persons;
```

Result

LastName	FirstName	City
Perera	Onali	Colombo
Herath	Chaminda	Kandy
Weerasekara		Galle

The Result Set

The result from a SQL query is stored in a result-set. Most database software systems allow navigation of the result set with programming functions, like Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc.

Task 13: Using SQL WHERE Clause

The WHERE clause is used to specify a selection criterion to conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

Syntax

```
SELECT column FROM table
WHERE column operator value
```

Relational operators in SQL Statements

With the WHERE clause, the following relational operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN / NOT IN	To specify multiple possible values for a column

Note: In some versions of SQL the <> operator may be written as !=

To select only the persons living in the city "Colombo", we add a WHERE clause to the SELECT statement:

```
SELECT * FROM Persons
WHERE City='Colombo';
```

If "Persons" table is

LastName	FirstName	City
Perera	Onali	Colombo
Herath	Chaminda	Kandy
Weerasekara		Galle

The result is:

LastName	FirstName	City
Perera	Onali	Colombo

Task 14: Using the 'Between'

E.g. If you want to list all the persons who are within the age range of 30 to 50, what the operator that we can use is?

```
SELECT *
FROM Persons
WHERE age BETWEEN 30 AND 50;
```

Task 15: Using the LIKE Condition

The LIKE condition is used to specify a search for a pattern in a column.

Syntax

```
SELECT column FROM table
WHERE column LIKE pattern;
```

A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%' ;
```

The following SQL statement will return persons with first names that end with an 'i':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%i' ;
```

The following SQL statement will return persons with first names that contain the pattern 'li':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%li%';
```

Task 16: Using the IN/NOT IN

E.g. If you want to select Persons living in cities other than Colombo and Kandy you may use the Not IN operator.

```
SELECT *
FROM Persons
WHERE city NOT IN('Colombo', 'Kandy');
```

Using Quotes

Note that we have used single quotes around the conditional values in the examples.

SQL uses single quotes around text values (most database systems will also accept double quotes).

Numeric values should not be enclosed in quotes.

```
This is correct:
SELECT * FROM Persons WHERE FirstName='Onali';
This is wrong:
SELECT * FROM Persons WHERE FirstName=Onali;
```

For numeric values:

```
This is correct:
SELECT * FROM Persons WHERE age>25;
This is wrong:
SELECT * FROM Persons WHERE age>'25';
```

Arithmetic operators in SQL Statements

Task 17: Using arithmetic operators in SQL

SQL commands are often used in conjunction with arithmetic operators. As you perform mathematical operations on attributes, remember the rules of precedence.

1. Perform operations within parentheses

2. Perform power operations

3. Perform multiplications and divisions

4. Perform additions and subtractions

Consider the following table “Ticket”

PARK_CODE	TICKET_NO	TICKET_TYPE	TICKET_PRICE	T
SP4533	11001	Adult	24.99	
SP4533	11002	Child	14.99	
SP4533	11003	Senior	10.99	
FR1001	13001	Child	18.99	
FR1001	13002	Adult	34.99	
FR1001	13003	Senior	20.99	
ZA1342	67832	Child	18.56	
ZA1342	67833	Adult	28.67	
ZA1342	67855	Senior	12.12	
UK3452	88567	Child	22.50	
UK3452	88568	Adult	42.10	
UK3452	89720	Senior	10.99	

Suppose the owners of all the theme parks wanted to compare the current ticket prices, with an increase in the price of each ticket by 10%.

You’re required to show all the ticket details (including the new ticket price) in the Ticket table. But remember we are not storing the new ticket values in the table.

Now try the following code snippets and compare the outcomes.

```
SELECT PARK_CODE, TICKET_NO, TICKET_TYPE, TICKET_PRICE,
TICKET_PRICE + ROUND((TICKET_PRICE *0.1),2)

FROM TICKET;
```

```
SELECT PARK_CODE, TICKET_NO, TICKET_TYPE, TICKET_PRICE, TICKET_PRICE
+ ROUND((TICKET_PRICE *0.1),2) PRICE_INCREASE

FROM TICKET;
```

```
SELECT PARK_CODE, TICKET_NO, TICKET_TYPE, TICKET_PRICE, TICKET_PRICE
+ ROUND((TICKET_PRICE *0.1),2) AS
“PRICE INCREASE”

FROM TICKET;
```

Logical Operators in SQL Statements

SQL allows you to have multiple conditions in a query through the use of logical operators: AND, OR and NOT. NOT has the highest precedence, followed by AND, and then followed by OR. However, you are strongly recommended to use parentheses to clarify the intended meaning of the query.

Original Table (used in the examples)

LastName	FirstName	City
Perera	Onali	Colombo
Herath	Chaminda	Kandy
Weerasekara	Manoja	Galle

Task 18: Using AND Operator

This logical AND connective is used to set up a query where there are two conditions which must be met for the query to return the required row(s)

Syntax: AND , &&

E.g. Use AND to display each person with the first name equal to "Manoja", and the last name equal to " Weerasekara ":

```
SELECT * FROM Persons
WHERE FirstName='Manoja'
AND LastName='Weerasekara';
```

Result:

LastName	FirstName	City
Weerasekara	Manoja	Galle

Task 19: Using OR Operator

The OR allows you to combine two Boolean expressions. It returns TRUE when either of the conditions evaluates to TRUE.

Syntax: OR , ||

E.g. Use OR to display each person with the first name equal to "Manoja", or the last name equal to "Herath":

```
SELECT * FROM Persons
WHERE firstname='Manoja'
OR lastname='Herath';
```

Result:

LastName	FirstName	City
Herath	Chaminda	Kandy
Weerasekara	Manoja	Galle

You can also combine AND and OR (use parentheses to form complex expressions):

```
SELECT * FROM Persons WHERE
(FirstName='Manoja' OR FirstName='Onali')
AND LastName='Weerasekara';
```

Result:

LastName	FirstName	City
Weerasekara	Manoja	Galle

Task 20: Using NOT Operator

The logical operator NOT is used to negate the result of a conditional expression.

Syntax: NOT , !

E.g. Write a query to display the a listing of all rows for which id is not 10060 from Persons table

```
SELECT * FROM Persons WHERE !(Id=10060);

You may write it as

SELECT * FROM Persons WHERE NOT(Id=10060);
```

Sorting Data

Task 21: Using ORDER BY clause

The ORDER BY clause is especially useful when the listing order of the query is important. Although you have the option of declaring the order type—ascending (ASC) or descending (DESC) —the default order is ascending.

Original Table:

Company	OrderNumber
Singer	3412
Beko	5678
Panasonic	2312
Panasonic	6798

Example

To display the companies in alphabetical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company
```

Result:

Company	OrderNumber
Beko	5678
Singer	3412
Panasonic	6798
Panasonic	2312

Example

To display the companies in alphabetical order AND the ordernumbers in numerical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

Result:

Company	OrderNumber
Beko	5678
Panasonic	2312
Panasonic	6798
Singer	3412

Example

To display the companies in reverse alphabetical order:

```
SELECT Company FROM Orders
ORDER BY Company DESC
```

Result:

Company
Singer
Panasonic
Panasonic
Beko

Basic SQL Aggregate Functions

Task 22: Using aggregate functions in a SQL query

There are several basic types and categories of functions in SQL. The basic types of functions are:

- Aggregate Functions
- Scalar functions (We'll discuss this later)

Aggregate functions operate against a collection of values but return a single value.

Note: If used among many other expressions in the item list of a SELECT statement, the SELECT must have a GROUP BY clause!!

Following table shows different aggregate functions and how they are used in a SQL query.

Function Name	Purpose	Example
COUNT	The number of rows containing non-null values	SELECT COUNT(PARK_CODE) FROM ATTRACTION;
DISTINCT	Produce a list of only those values that are different from one another ****Not an Aggregate function**	SELECT DISTINCT(PARK_CODE) FROM ATTRACTION; SELECT COUNT(DISTINCT(PARK_CODE)) FROM ATTRACTION;
MIN	The minimum attribute value encountered in a given column	SELECT MIN(TICKET_PRICE) FROM TICKET;
MAX	The maximum attribute value encountered in a given column	SELECT max(TICKET_PRICE) FROM TICKET;
SUM	The sum of all values for a given column	SELECT SUM(LINE_QTY) FROM SALES_LINE;

AVG	The arithmetic mean (average) for a specified column	SELECT AVG(LINE_PRICE) FROM SALES_LINE;

Task 23: How to use Group By clause in a SQL query

The GROUP BY clause is generally used when you have attribute columns combined with aggregate functions in the SELECT statement.

It is valid only when used in conjunction with one of the SQL aggregate functions, such as COUNT, MIN, MAX, AVG and SUM.

The GROUP BY clause appears after the WHERE statement.

When using GROUP BY you should include all the attributes that are in the SELECT statement that do not use an aggregate function.

This "Sales" Table:

Company	Amount
Singer	5500
Panasonic	4500
Singer	7100

And This SQL:

```
SELECT Company, SUM(Amount) FROM Sales
```

Returns this result:

Company	SUM(Amount)
Singer	17100
Panasonic	17100
Singer	17100

The above code is invalid because the column returned is not part of an aggregate. A GROUP BY clause will solve this problem:

```
SELECT Company, SUM(Amount) FROM Sales
GROUP BY Company
```

Returns this result:

Company	SUM(Amount)
Singer	12600
Panasonic	4500

Task 24: How to use Having clause in a SQL query

HAVING... was added to SQL because the WHERE keyword could not be used against aggregate functions (like SUM), and without HAVING... it would be impossible to test for result conditions.

The syntax for the HAVING function is:

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

This "Sales" Table:

Company	Amount
Singer	5500
Panasonic	4500
Singer	7100

This SQL:

```
SELECT Company, SUM(Amount) FROM Sales
GROUP BY Company
HAVING SUM(Amount)>10000
```

Returns this result

Company	SUM(Amount)
Singer	12600

SQL ALTER TABLE Statement

Task 25: How to use Alter Table in SQL statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table. The ALTER

TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE – ADD/ DROP/MODIFY Column

```
ALTER TABLE table_name
ADD column_name datatype;
```

```
ALTER TABLE table_name
DROP COLUMN column_name;

ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

Note: Some database systems don't allow the dropping of a column in a database table (DROP COLUMN column_name).

Person:

LastName	FirstName	Address
Pettersen	Kari	Storgt 20

Example

To add a column named "City" in the "Person" table:

```
ALTER TABLE Person ADD City varchar(30)
```

Result:

LastName	FirstName	City	Address
Perera	Onali	Colombo	Hudson Rd

Example

To drop the "Address" column in the "Person" table:

```
ALTER TABLE Person DROP COLUMN Address
```

Result:

LastName	FirstName	City
Perera	Onali	Colombo

The SQL UPDATE Statement

Task 26: How to use UPDTAE statement in SQL

The UPDATE statement is used to modify the existing records in a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

```
UPDATE    Customers
SET ContactName='Juan';
```

This is wrong! This will update all the conatctNames to Juan ☺

How to Back Up MySQL Databases from the Command Line

To dump/export a MySQL database, execute the following command in the Windows command prompt: mysqldump -u username -p dbname > filename.sql.

After entering that command, you will be prompted for your password. Once the password is entered your dump file will be available in the root directory for your Windows user – ie: C:\Users\username.

But what is a dump?

SQL Quick Reference

Statement	Syntax
AND / OR	SELECT column_name(s) FROM table_name WHERE condition AND|OR condition
ALTER TABLE (add column)	ALTER TABLE table_name ADD column_name datatype
ALTER TABLE (drop column)	ALTER TABLE table_name DROP COLUMN column_name
AS (alias for column)	SELECT column_name AS column_alias FROM table_name
AS (alias for table)	SELECT column_name FROM table_name AS table_alias
BETWEEN	SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2
CREATE DATABASE	CREATE DATABASE database_name
CREATE INDEX	CREATE INDEX index_name ON table_name (column_name)
CREATE TABLE	CREATE TABLE table_name (column_name1 data_type, column_name2 data_type, ...)
CREATE UNIQUE INDEX	CREATE UNIQUE INDEX index_name ON table_name (column_name)
CREATE VIEW	CREATE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition
DELETE FROM	DELETE FROM table_name (**Note:** Deletes the entire table!!) *or* DELETE FROM table_name WHERE condition
DROP DATABASE	DROP DATABASE database_name
DROP INDEX	DROP INDEX table_name.index_name
DROP TABLE	DROP TABLE table_name
GROUP BY	SELECT column_name1,SUM(column_name2) FROM table_name GROUP BY column_name1

HAVING	SELECT column_name1,SUM(column_name2) FROM table_name GROUP BY column_name1 HAVING SUM(column_name2) condition value
IN	SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,..)
INSERT INTO	INSERT INTO table_name VALUES (value1, value2,....) *or* INSERT INTO table_name (column_name1, column_name2,...) VALUES (value1, value2,....)
LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern
ORDER BY	SELECT column_name(s) FROM table_name ORDER BY column_name [ASC|DESC]
SELECT	SELECT column_name(s) FROM table_name
SELECT *	SELECT * FROM table_name
SELECT DISTINCT	SELECT DISTINCT column_name(s) FROM table_name
SELECT INTO (used to create backup copies of tables)	SELECT * INTO new_table_name FROM original_table_name *or* SELECT column_name(s) INTO new_table_name FROM original_table_name
TRUNCATE TABLE (deletes only the data inside the table)	TRUNCATE TABLE table_name
UPDATE	UPDATE table_name SET column_name=new_value [, column_name=new_value] WHERE column_name=some_value
WHERE	SELECT column_name(s) FROM table_name WHERE condition

Though this is the end of the SQL discussions related to your level there are so many other things you can refer to and learn ☺

Now it's your turn to try. Following is complete task that you can do to check whether you understood what we have learnt so far…

Let's Try

Task 01:

Consider the below class diagram developed using ER data.

Cutomer	**Invoice**
Cus_code (int)-PK	Inv_No (int)-PK
Cus_name (varchar)-Not Null	Cus_code (int)-FK
Cus_balance(double)-Not null	Inv_amount(double)-Not null

Try to develop above tables in the database called "SALES". Now try to insert records as below:

```
INSERT INTO customer VALUES(1,'Mark',500.00),(2,'Bogdan',100.00);
INSERT INTO invoice VALUES(1,1,100.00),(3,2,200.00);
```

I. What is the output?

II. Now try to insert records as below:
```
INSERT INTO invoice VALUES(2,3,300.00);
```

III. What is the output? Justify your answer.

IV. USE **SALES** DB and **Customer** table. Add new column as below.

Cutomer

Cus_code (int)-PK

Cus_name (varchar)-Not Null

Cus_balance(double)-Not null

Cus_city (char)

V. Then try to change the data type of the Cus_city to varchar.

VI. Display the properties of the table. <find out how to do this>

VII. Now try to remove the newly added column from the table.

VIII. USE **SALES** and create the following tables.

ITEM

SalesID-INT - PK
OrderDescription –VARCHAR-NOT NULL

Quantity- INT- NOT NULL
Price- INT- NOT NULL
ItemDescription–VARCHAR

IX. Insert following details to the table.

	ITEM			
SalesID	100	200	300	400
OrderDescription	Frock	Trouser	T-shirts	Shirts
Quantity	20	15	30	25
Price	1200	1850	1300	2000
ItemDescription	New with tag		New with tag	

X. Now update the price of the item to Rs.1500 of the salesID 300.

XI. Now add an ItemDescription to salesID 400

XII. Write a query to finds all rows from ITEMS where Order Description begin with the letter *T*.

XIII. Write a query to finds all rows from ITEMS where Order Description ends with the letter *S*.

XIV. Write a query to finds all rows from ITEMS where Order Description contains “ir” in any place.

XV. Write a query to finds all rows from ITEMS where Order Description starts with “F” and ends with “S”.

XVI. Now try to dump “SALES” database.

XVII. Find out how to restore a DB using CMD ☺

Task 02:

I. Creative Visual Company maintains a DB called “Movies”. This DB contains following 2 tables with the given specification.

Table 1: Movie_Titles

MovieID- INT-PK

MovieName- Varchar- 45-This cannot be Null

MovieRatingLevel-INT- This cannot be Null (E.g. 1,2,3,4, or 5)

MovieReleaseDate- Date-This can be Null

Table 2: Movie_Display

MovieHallID-INT-PK

MovieID- INT-FK

MovieHallName- Varchar- 20-This cannot be Null

MovieHallLocation- Varchar- 25-This can be Null

MovieTicketPrice-INT- This cannot be Null

II. Insert 3 records to each table.

III. Movie hall owners planned to increase the ticket price by 10%. You're required to add a new column to the Movie_Display table called "MovieTicketNEWPrice" and store the new value of the ticket.

IV. Write an SQL query to fetch records from Movie_Display table where movie hall is in following cities: Colombo, Kandy, Galle, Kurunegala.

V. Write an SQL query to fetch records from Movie_Titles table where movie is directed by "Christopher Nolan" and rated above level 5.

VI. Now try to delete Movie_Display table from the DB. What is the putput?

Task 03:

I. Now create the SALES database in MySQL.

II. USE SALES and create the following tables.

SALES

SalesID-INT - PK
OrderDescription –VARCHAR-NOT NULL
Quantity- INT- NOT NULL
Price- INT- NOT NULL
ItemDescription–VARCHAR

III. Insert following details to the table.

	ITEM			
SalesID	100	200	300	400
OrderDescription	Frock	Trouser	T-shirts	Shirts
Quantity	20	15	30	25
Price	1200	1850	1300	2000

ItemDescription	New with tag	New with tag

Now develop queries to retrieve following information.

IV. Display all the details of the SALES table.

V. Display only the SalesID and Sales description of all the records.

VI. Display all details of the item where SalesID is equals to 200.

VII. Display only Sales description of item where SalesID is equals to 200.

VIII. Shop owner planned to increase all the prices of the items by 10%. Display all details of the items along with price increment.

IX. Try to execute the same by renaming the column header of the temporary column to "NEWPRICE".

X. Display Sales IDs where price is less than Rs.1500.

XI. Display Order Description of items where price is above 1500 but less than 2000.

XII. Assume there are many other records in the same SALES table. You are required to write a query to filter records where their Order Description is not Frock, Trouser, T-shirts or Shirts.

XIII. Display Order Description of items where their price is above 1500 and available quantity is above 20.

XIV. Display Order Description of items where their price is above 1500 or available quantity is less than 15.

XV. Display Order Description of items where their price is not equals 1500.

XVI. Display all items listed by Price in descending order.

XVII. Display Order Description of items where their price is above 1500 and available quantity is above 20 listed by Price in ascending order.

XVIII. Display the Order Description of the item where its price is the lowest in the list.

XIX. Display the Order Description of the item where its availability is the highest in the list.

XX. Display the average price of the items available in the SALES table.

Let's discuss the answers during the laboratory session ☺

------END------