Ahsanullah University of Science and Technology (AUST)

Department of Computer Science and Engineering

LABORATORY MANUAL

Course No. : CSE 3104
Course Title: Database Lab

For the students of 3rd Year, 1st semester of
B.Sc. in Computer Science and Engineering program

TABLE OF CONTENTS

COURSE OBJECTIVES

1. Introduction to database systems concept including database querying, design, and programming.
2. Design and implement normalized database structures by creating simple database tables, queries, procedures, Entity- Relationship (ER) diagrams.
3. Upon completion, students should be able to design and implement current database technologies, and to use and develop database associated applications.

PREFFERED TOOL(S)

- MySQL
- Net Beans

TEXT/REFERENCE BOOK(S)

- "SQL The Complete Reference written by James R. Groff, Paul N. Weinberg, Andrew J. Oppel"
- "Head First SQL written by Lynn Beighley"
- "Database System Concepts" written by 'Abraham Silberschatz' 'Henry F. Korth' 'S. Sudershan'.

ADMINISTRATIVE POLICY OF THE LABORATORY

- ✓ Students must be performed class assessment tasks individually without help of others.
- ✓ Viva for each program will be taken and considered as a performance.
- ✓ Plagiarism is strictly forbidden and will be dealt with punishment.

Session 1

Objective: Getting acquainted with SQL and MySQL Server.

1 Introduction to SQL

SQL (Structured Query Language) is a database computer language designed for managing data in relational database management systems (RDBMS).

SQL is a standardized computer language that was originally developed by IBM for querying, altering and defining relational databases, using declarative statements.

What can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database

SQL can create views in a database Even if SQL is a standard, many of the database systems that exist today implement their own version of the SQL language. In this document we will use the MySQL Server as an example.

There are lots of different database systems, or DBMS–Database Management Systems, such as:

- Microsoft SQL Server
- Oracle
- MySQL (Oracle, previously Sun Microsystems)- MySQL can be used free of charge (open source license), Web sites that use MySQL: YouTube, Wikipedia, Facebook
- Microsoft Access
- IBM DB2
- Sybase
- lots of other systems

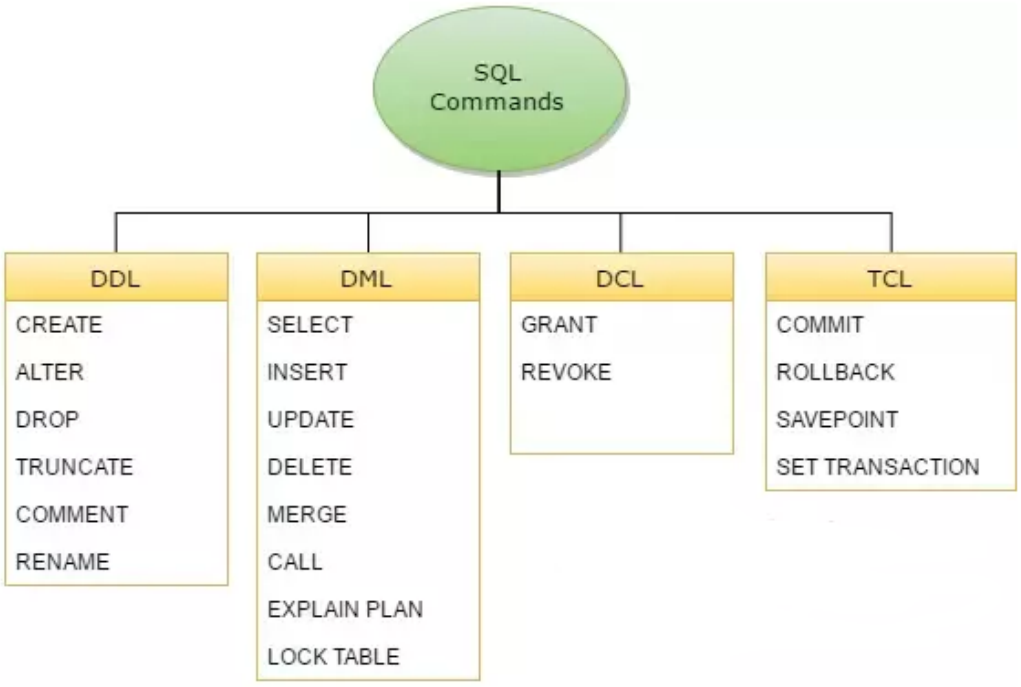
1.1 Data Definition Language (DDL)

A data definition language or data description language (DDL) is a syntax similar to a computer programming language for defining data structures, especially database schemas**.**

DDL statements create, modify, and remove database objects such as tables, indexes, and users. Common DDL statements are CREATE, ALTER, and DROP.

1.2 Data Manipulation Language (DML)

The DML section is used to manipulate the data such as querying it. While is also common to use a query builder to create queries, people do still hand-craft DML statements, such as queries.



2 Introduction to MySQL

MySQL Enterprise Edition includes the most comprehensive set of advanced features, management tools and technical support to achieve the highest levels of MySQL scalability, security, reliability, and uptime. It reduces the risk, cost, and complexity in developing, deploying, and managing business-critical MySQL applications.

MySQL is an open source relational database management system (RDBMS) based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX, and Windows. Although it can be used in a wide range of applications, MySQL is most often associated with web-based applications and online publishing and is an important component of an open source enterprise stack called LAMP. LAMP is a Web development platform that uses Linux as the operating system, Apache as the Web server, MySQL as the relational database management system and PHP as the object-oriented scripting language. (Sometimes Perl or Python is used instead of PHP.)

2.1 Commands for establishing connection with MySQL

1. cd/
2. c:
3. cd xampp
4. cd mysql
5. cd bin
6. mysql –u root –p –h

Session 2

Objective: Creating and Accessing Database, Creating Tables, Loading and Retrieving commands, Altering, Renaming SQL Commands.

1 Creating and Accessing a Database

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to perform the following operations:

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL website. Use the SHOW statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
```

The mysql database describes user access privileges. The test database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; SHOW DATABASES does not show databases that you have no privileges for if you do not have the SHOW DATABASES privilege.

If the test database exists, try to access it:

```
mysql> USE test
Database changed
```

USE, like QUIT, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The USE statement is special in another way, too: it must be given on a single line.

2 Creating a Table

```
CREATE TABLE CUSTOMER
(
CustomerId int auto_increment PRIMARY KEY,
LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
AreaCode int NULL,
Address varchar(200) NULL,
Phone varchar(11) NULL
);
```

2.1 Data Types in MySQL

In MySQL there are three main types : text, number, and Date/Time types.

Text types:

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
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data

MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. 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. **Note:** The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Number types:

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
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

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

Date types:

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:MM: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:MM: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:MM: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

Note: Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also accepts various formats, like YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, or YYMMD.

3 Loading Data Into Table

```
INSERT INTO CUSTOMER VALUES (1000, 'Rahman', 'Karim', 1203,
'Dhaka','01912584949');
INSERT INTO CUSTOMER VALUES ('Rahman', 'Karim', 1203,
'Dhaka','01912584949');

INSERT INTO CUSTOMER (LastName, FirstName, AreaCode, Address, Phone)
VALUES ('Khan', 'Rahim', 1307, 'Gulshan', '01677515829');

INSERT INTO CUSTOMER (LastName, FirstName)
VALUES ('Ahmed', 'Hashim');
```

4 Altering Tables

```
ALTER TABLE CUSTOMER
ADD CustomerSince date;

ALTER TABLE CUSTOMER
MODIFY COLUMN CustomerSince datetime;

ALTER TABLE CUSTOMER
DROP COLUMN CustomerSince;
```

5 Retrieving Information from a Table

```
SELECT * FROM CUSTOMER;
```

6 Truncate Command

```
TRUNCATE TABLE TABLE_NAME;
```

Session 3

Objective: Learning about SQL Constraints.

1 SQL Constraints

Constraints can be specified when a table is being created (with create command) or after the table is created.

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

1.1 Primary Key

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values. When multiple fields are used as a primary key, they are called a composite key.

1. The PRIMARY KEY constraint uniquely identifies each record in a database table.
2. Primary keys must contain UNIQUE values, and cannot contain NULL values.
3. A table can have only one primary key, which may consist of single or multiple fields.

```
CREATE TABLE CUSTOMER
(
CustomerId int PRIMARY KEY,
LastName varchar(50),
FirstName varchar(50),
AreaCode int,
Address varchar(200),
Phone varchar(11),
);
```

1.2 Foreign Key

SQL Foreign Key. A foreign key is a column (or columns) that references a column (most often the primary key) of another table. The purpose of the foreign key is to ensure referential integrity of the data. In other words, only values that are supposed to appear in the database are permitted.

1. A FOREIGN KEY is a key used to link two tables together.
2. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
3. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

```
CREATE TABLE customer
(
CustomerId int auto_increment PRIMARY KEY,
CustomerNumber int NOT NULL UNIQUE  CHECK(CustomerNumber>1000),
LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
AreaCode int NULL,
Address varchar(200) NULL  DEFAULT  'Dhaka',
Phone varchar(11) NULL
);


CREATE TABLE `order`
(
OrderId int auto_increment,
CustomerId int NOT NULL,
OrderDate Date,
OrderAmount double(7,2),
PRIMARY KEY(OrderId),
FOREIGN KEY(CustomerId) REFERENCES customer(CustomerId)
);
```

1.3 AUTO_INCREMENT

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. Often this is the primary key field that we would like to be created automatically every time a new record is inserted. MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature. By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

```
INSERT INTO CUSTOMER (CustomerNumber,LastName, FirstName, AreaCode,
Address, Phone)
VALUES (1001,'Khan', 'Rahim', 1307, 'Gulshan', '01677515829'),
       (1002,'Rahman', 'Kahim', 1202, 'Dhanmondi', '01912584949'),
       (1003,'Mehedi', 'Hashim', 1307, 'Gulshan', '01645789544');
```

```
ALTER TABLE customer AUTO_INCREMENT=100;

INSERT INTO CUSTOMER (CustomerNumber,LastName, FirstName, AreaCode,
Address, Phone)
VALUES (1004,'Sardar', 'Kashim', 1102, 'Uttara', '01612457845');

INSERT INTO CUSTOMER (CustomerNumber,LastName, FirstName, AreaCode, Phone)
VALUES (1005,'Sardar', 'Kashim', 1107, '01612357845');
```

1.4 DEFAULT

The DEFAULT constraint is used to provide a default value for a column.The default value will be added to all new records IF no other value is specified.

```
CREATE TABLE customer
(
CustomerId int auto_increment PRIMARY KEY,
CustomerNumber int NOT NULL UNIQUE CHECK(CustomerNumber>1000),
LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
AreaCode int NULL,
Address varchar(200) NULL  DEFAULT  'Dhaka',
Phone varchar(11) NULL
);
```

1.5 NOT NULL

The UNIQUE constraint ensures that all values in a column are different. Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns. A PRIMARY KEY constraint automatically has a UNIQUE constraint. However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

```
CREATE TABLE customer
(
CustomerId int auto_increment PRIMARY KEY,
CustomerNumber int NOT NULL UNIQUE CHECK(CustomerNumber>1000),
LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
AreaCode int NULL,
Address varchar(200) NULL  DEFAULT  'Dhaka',
Phone varchar(11) NULL
);
```

Session 4

Objective: Learning about Update, Select, Delete Statements.

1 UPDATE Statements

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

For Updating values in a table, we first need to create a table and load some values in the table.

```
CREATE TABLE customer
(
CustomerId int auto_increment PRIMARY KEY,
CustomerNumber int NOT NULL UNIQUE  CHECK(CustomerNumber>1000),
LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
AreaCode int NULL,
Address varchar(200) NULL  DEFAULT  'Dhaka',
Phone varchar(11) NULL
);
```

```
INSERT INTO CUSTOMER (CustomerNumber,LastName, FirstName, AreaCode,
Address, Phone)
VALUES (1001,'Khan', 'Rahim', 1307, 'Gulshan', '01677515829'),
       (1002,'Rahman', 'Kahim', 1202, 'Dhanmondi', '01912584949'),
       (1003,'Mehedi', 'Hashim', 1307, 'Gulshan', '01645789544');

ALTER TABLE customer AUTO_INCREMENT=100;

INSERT INTO CUSTOMER (CustomerNumber,LastName, FirstName, AreaCode,
Address, Phone)
VALUES (1004,'Sardar', 'Kashim', 1102, 'Uttara', '01612457845');

INSERT INTO CUSTOMER (CustomerNumber,LastName, FirstName, AreaCode, Phone)
VALUES (1005,'Sardar', 'Kashim', 1107, '01612357845');
```

```
UPDATE CUSTOMER set AreaCode=46 where CustomerId=2;
UPDATE CUSTOMER set AreaCode=46;
```

2 DELETE RECORDS FROM TABLE

The DELETE statement is used to delete records from a table:

```
DELETE FROM table_name
WHERE some_column = some_value
```

```
DELETE FROM CUSTOMER where CustomerId=2;
```

3 SELECT Statements

The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set. The SELECT statement is used to select data from one or more tables:

```
SELECT column_name(s) FROM table_name;
SELECT * FROM CUSTOMER;
SELECT CustomerId, FirstName, LastName FROM CUSTOMER;
```

3.1 Aliasing

The WHERE clause is used to filter records. The WHERE clause is used to extract only those records that fulfill a specified condition.

```
SELECT FirstName as 'First Name',LastName as 'Last Name' FROM CUSTOMER;
```

3.2 Sorting Records

The ORDER BY keyword is used to sort the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT * FROM CUSTOMER ORDER BY LastName;
SELECT * FROM CUSTOMER ORDER BY Address,LastName;
SELECT * FROM CUSTOMER ORDER BY LastName desc;
```

3.3 SELECT Statement with WHERE clause

```
SELECT * FROM CUSTOMER WHERE CustomerNumber='1001';
SELECT * FROM CUSTOMER WHERE AreaCode > 1200;
```

3.4 DISTINCT keyword

The SELECT DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values. The SELECT DISTINCT statement is used to return only distinct (different) values.

```
SELECT DISTINCT Address FROM CUSTOMER;
```

3.5 LIMIT Data Selections

MySQL provides a LIMIT clause that is used to specify the number of records to return.The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.

```
SELECT * FROM CUSTOMER
LIMIT 2;
```

Session 5

Objective: Learning about Operators used with WHERE clause.

1 OPERATORS

With the WHERE clause, the following 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	If you know the exact value you want to return for at least one of the columns

1.1 LIKE operator

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

There are two wildcards used in conjunction with the LIKE operator:

"%" The percent sign represents zero, one, or multiple characters

"_" The underscore represents a single character

Examples:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

1.2 IN Operator

The IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);

OR,

SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);

SELECT * FROM CUSTOMER
WHERE Address IN('GULSHAN','DHAKA');
```

1.3 BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;

SELECT * FROM CUSTOMER

WHERE CustomerID BETWEEN 1 AND 100;
```

1.4 AND, OR and NOT Operator

The WHERE clause can be combined with AND, OR, and NOT operators.The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND is TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Session 6

Objective: Developing Entity Relationship Diagrams (ERD).

Why ERDs?

Entity Relationship Diagrams are a major data modeling tool and will help organize the data in your project into entities and define the relationships between the entities. This process has proved to enable the analyst to produce a good database structure so that the data can be stored and retrieved in a most efficient manner.

By using a graphical format it may help communication about the design between the designer and the user and the designer and the people who will implement it.

Components of an ERD

An ERD typically consists of four different graphical components:

- **Entity. (Nouns)**

  A data entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things or concepts.E.g. employee, payment, campus, book. Specific examples of an entity are called **instances**. E.g. the employee Karim Rahman, Hashim Khan's payment, etc.

- **Relationship. (Verb)**

  A data relationship is a natural association that exists between one or more entities. E.g. Employees ***process*** payments.

- **Cardinality. (Adverbs)**

  Defines the number of occurrences of one entity for a single occurrence of the related entity. E.g. an employee may process many payments but might not process any payments depending on the nature of her job.

- **Attribute. (Adjective , But they are often Nouns)**

  A data attribute is a ***characteristic*** common to all or most instances of a particular entity. Synonyms include property, data element, field. E.g. Name, address, Employee Number, pay rate are all attributes of the entity employee. *An attribute or combination of attributes that uniquely identifies one and only one instance of an entity is called a primary key or identifier*. E.g. Employee Number is a primary key for Employee.

One Methodology for Developing an ERD

Typically you will start with a ***case study*** or perhaps a ***logical model*** of the system to be developed. This document will demonstrate how to use the following process to convert that information into an ERD.

The process has ten steps:

1. Identify Entities

Identify the roles, events, locations, tangible things or concepts about which the end-users want to store data.

2. Find Relationships

Find the natural associations between pairs of entities using a relationship matrix.

3. Draw Rough ERD

Put entities in rectangles and relationships on line segments connecting the entities.

4. Fill in Cardinality

Determine the number of occurrences of one entity for a single occurrence of the related entity.

5. Define Primary Keys

Identify the data attribute(s) that uniquely identify one and only one occurrence of each entity.

6. Draw Key-Based ERD

Eliminate Many-to-Many relationships and include primary and foreign keys in each entity.

7. Identify Attributes

Name the information details (fields) which are essential to the ***system under development***.

8. Map Attributes

For each attribute, match it with exactly one entity that it describes.

9. Draw fully attributed ERD

Adjust the ERD from step 6 to account for entities or relationships discovered in step 8.

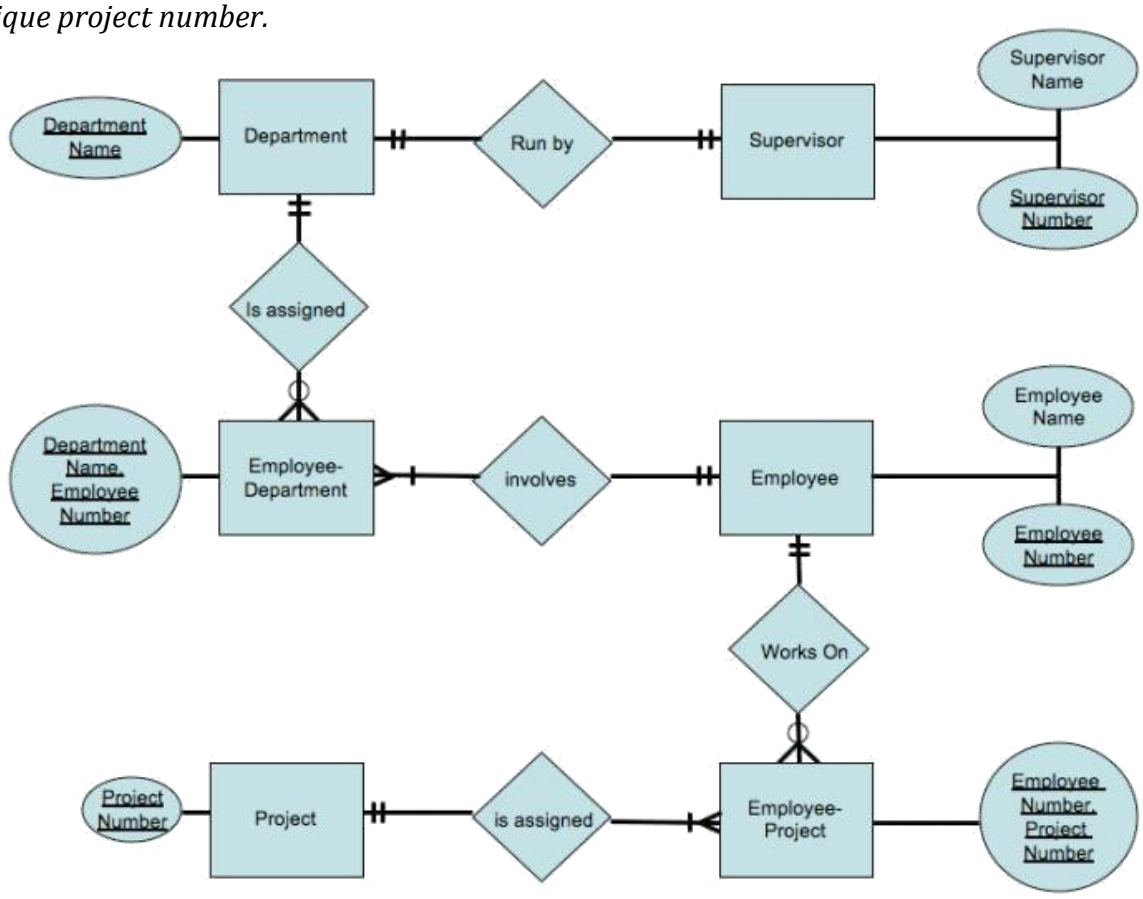
10. Check Results

Does the final Entity Relationship Diagram accurately depict the system data?

A Simple Example

The above process will be illustrated by working through the following example.

A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to any projects. *The important data fields are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee number and a unique project number.*



Session 7

Objective: Converting ERD to Relational Model.

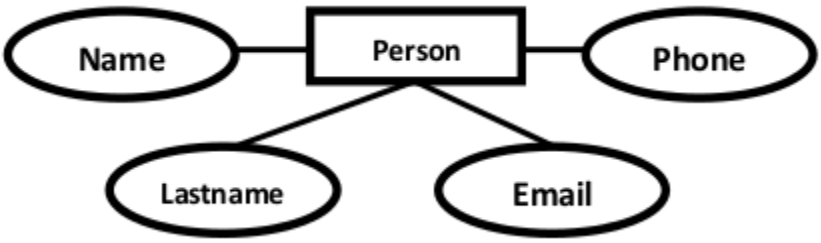
The ER Model is intended as a description of real-world entities. Although it is constructed in such a way as to allow easy translation to the relational schema model, this is not an entirely trivial process. The ER diagram represents the conceptual level of database design meanwhile the relational schema is the logical level for the database design. We will be following the simple rules:

1. Entities and Simple Attributes:

An entity type within ER diagram is turned into a table. You may preferably keep the same name for the entity or give it a sensible name but avoid DBMS reserved words as well as avoid the use of special characters.
Each attribute turns into a column (attribute) in the table. The key attribute of the entity is the primary key of the table which is usually underlined. It can be composite if required but can never be null.

Note: It is highly recommended that every table should start with its primary key attribute conventionally named as TablenameID.

Taking the following simple ER diagram:



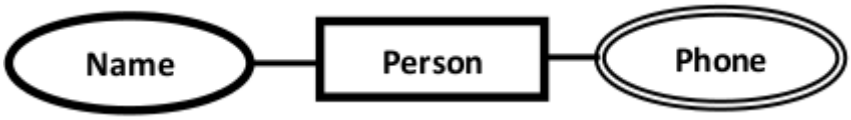
The initial relational schema is expressed in the following format writing the table names with the attributes list inside a parentheses as shown below for

Persons(<u>personid</u> , name, lastname, email)

Persons and Phones are Tables. name, lastname, are Table Columns (Attributes).

2. Multi-Valued Attributes

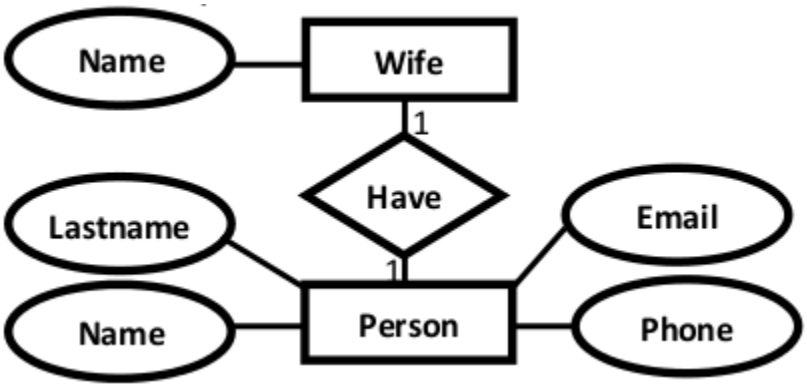
A multi-valued attribute is usually represented with a double-line oval.



If you have a multi-valued attribute, take the attribute and turn it into a new entity or table of its own. Then make a 1:N relationship between the new entity and the existing one. In simple words. 1. Create a table for the attribute. 2. Add the primary (id) column of the parent entity as a foreign key within the new table as shown below:

Persons(personid , name, lastname, email)
Phones (phoneid , ***personid***, phone)

3. 1:1 Relationships



To keep it simple and even for better performances at data retrieval, I would personally recommend using attributes to represent such relationship. For instance, let us consider the case where the Person has or optionally has one wife. You can place the primary key of the wife within the table of the Persons which we call in this case Foreign key as shown below.

Persons(personid , name, lastname, email , ***wifeid***)
Wife (wifeid , name)

Or vice versa to put the **personid** as a foreign key within the Wife table as shown below:

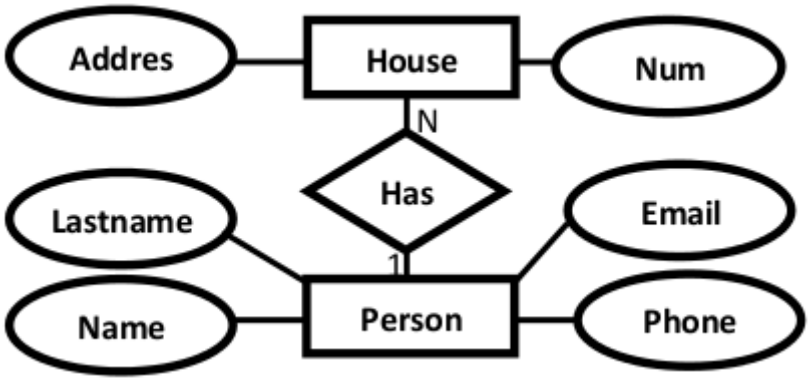
Persons(personid , name, lastname, email)
Wife (wifeid , name , ***personid***)

For cases when the Person is not married i.e. has no wifeID, the attribute can set to NULL.

4. 1:N Relationships

This is the tricky part!

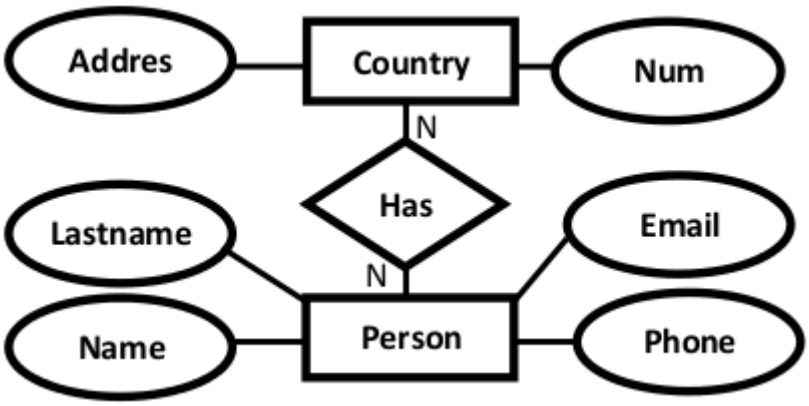
For simplicity, use attributes in the same way as 1:1 relationship but we have only one choice as opposed to two choices. For instance, the Person can have a **House** from zero to many, but a **House** can have only one **Person**. To represent such relationship the **personid** as the Parent node must be placed within the Child table as a foreign key but **not the other way around as shown next.**



It should convert to :
Persons(personid , name, lastname, email)
House (houseid , num , address, ***personid***)

5. N:N Relationships

We normally use tables to express such type of relationship. This is the same for N – ary relationship of ER diagrams. For instance, The Person can live or work in many countries. Also, a country can have many people. To express this relationship within a relational schema we use a separate table as shown below:



It should convert into:

Persons(personid , name, lastname, email)
Countries (countryid , name, code)
HasRelat (hasrelatid , **personid , countryid**)

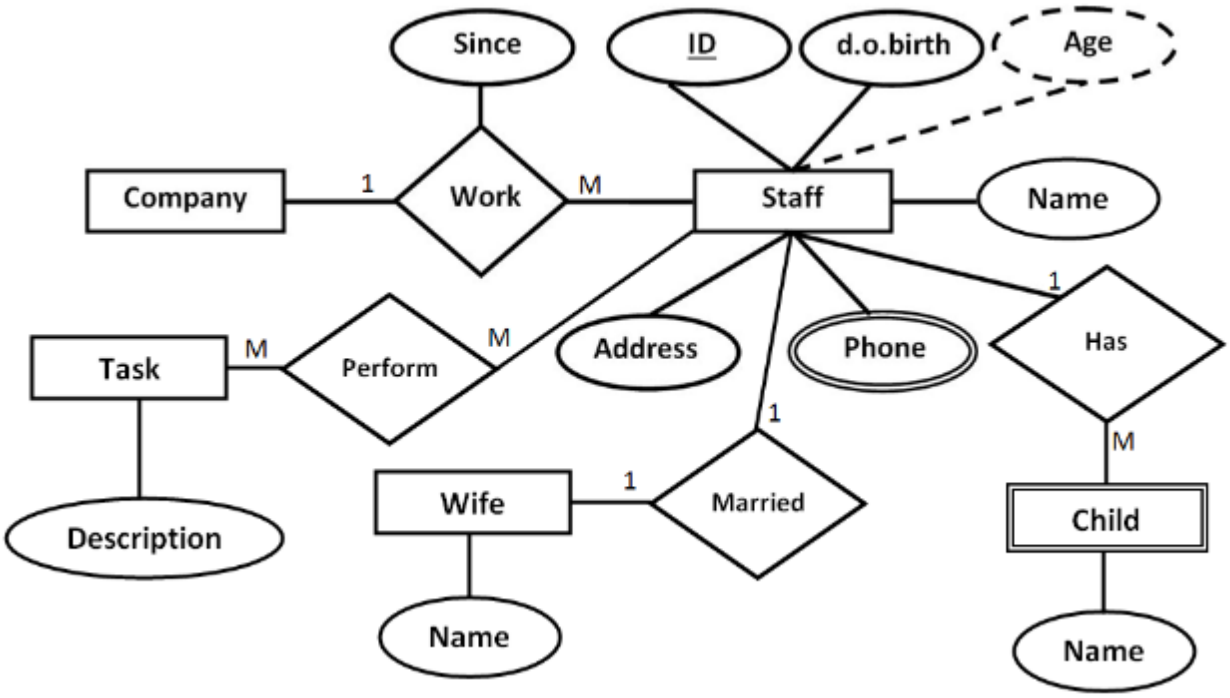
Relationship with attributes:

It is recommended to use table to represent them to keep the design tidy and clean **regardless of the cardinality** of the relationship.
Case Study

For the sake of simplicity, we will be producing the relational schema for the following ER diagram.

The relational schema for the ER Diagram is given below as:

Company(CompanyID , name , address)
Staff(StaffID , dob , address , ***WifeID***)
Child(ChildID , name , ***StaffID***)
Wife (WifeID , name)
Phone(PhoneID , phoneNumber , ***StaffID***)
Task (TaskID , description)
Work(WorkID , ***CompanyID , StaffID*** , since)
Perform(PerformID , ***StaffID , TaskID***)


Since
ID
d.o.birth
Age
Company
1
Work
M
Staff
Name
1
Address
Phone
Has
Task
M
Perform
M
1
M
Wife
1
Married
Child
Description
Name
Name

Session 8

Objective: Building a Java application with MySQL.

Let's try building a Java application with MySQL server.

IDE: Netbeans

Database Server: MySQL

Programming Language: Java

Application Requirements:

Develop an application with the following features:

- Takes a student's id, name and cgpa as input.
- Saves the information in a table.
- Updates information in a table.
- Deletes information from table.
- Searches information from table.

Connection MySQL Database with Java:

For connecting java application with the mysql database, you need to follow 5 steps to perform database connectivity. In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

A code fragment for connecting with database named XYZ with has been provided.

```
1. import java.sql.*;
2. class MysqlCon{
3. public static void main(String args[]){
4. try{
5. Class.forName("com.mysql.jdbc.Driver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:mysql://localhost:3306/XYZ","root","root");
```

To connect java application with the mysql database mysqlconnector.jar file is required to be loaded.

- download the jar file mysql-connector.jar

Two ways to load the jar file:

- paste the mysqlconnector.jar file in jre/lib/ext folder
- set classpath

1) paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2) set classpath:

There are two ways to set the classpath:

- temporary
- permanent

open command prompt and write to set temporary classpath:

C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

For setting permanent classpath, Go to environment variable then click on new tab. In variable name write classpath and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

Session 9

Objective: Learning about different types of aggregate functions.

1 AGGREGATE FUNCTIONS

The following are the most commonly used SQL aggregate functions:

- AVG – calculates the average of a set of values.
- COUNT – counts rows in a specified table or view.
- MIN – gets the minimum value in a set of values.
- MAX – gets the maximum value in a set of values.
- SUM – calculates the sum of values.

1.1 COUNT() Syntax:

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

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

1.2 AVG() Syntax:

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

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

1.3 SUM() Syntax:

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

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

1.4 MIN() Syntax:

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

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

1.5 MAX() Syntax:

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

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

2 GROUP BY Clause

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

3 The HAVING Clause

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

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

4 Some Related Examples

```
SELECT SUM(SALARY) as 'Total Salary',AVG(SALARY) as 'Average Salary' FROM
CUSTOMER;

SELECT MAX(SALARY)-MIN(SALARY) FROM CUSTOMER WHERE AGE >= 25;


SELECT COUNT(*) FROM CUSTOMER WHERE NAME LIKE 'Ka%';

SELECT Age, MAX(Salary) FROM CUSTOMER GROUP BY Age HAVING NAME LIKE 'Ka%';
SELECT Age, MAX(Salary) FROM CUSTOMER GROUP BY Age HAVING Age >= 25;
```

Session 10

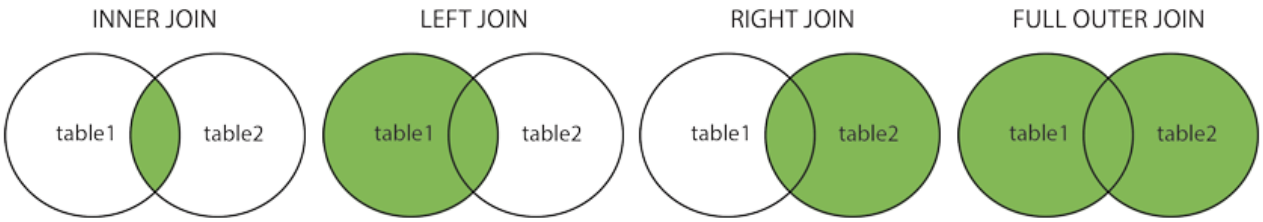
Objective: Learning about different types of SQL Join.

1 SQL Join

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table



1.1 Inner Join

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax:

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2* ON *table1.column_name* = *table2.column_name*;

```
SELECT C.CustomerId , Name, Age, O.Amount
FROM CUSTOMER C, ORDERS O
WHERE C.CustomerId = O.CustomerId;
```

1.2 SQL LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

LEFT JOIN Syntax:

SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2* ON *table1.column_name* = *table2.column_name*;

```
SELECT CUSTOMER.CustomerId, Name, Age, Amount,Date
FROM CUSTOMER
LEFT JOIN ORDERS
ON CUSTOMER.CustomerId = ORDERS.CustomerId;
```

1.3 SQL RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

RIGHT JOIN Syntax:

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2* ON *table1.column_name* = *table2.column_name*;

```
SELECT CUSTOMER.CustomerId , Name, Age, Amount,Date
FROM CUSTOMER
RIGHT JOIN ORDERS
ON CUSTOMER.CustomerId = ORDERS.CustomerId;
```

1.4 SQL FULL OUTER JOIN

The Full Outer Join return all records when there is a match in either left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

FULL OUTER JOIN Syntax:

SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2* ON *table1.column_name* = *table2.column_name*;

```
SELECT CUSTOMER.CustomerId, Name, Age, Amount,Date
FROM CUSTOMER
FULL OUTER JOIN ORDERS
ON CUSTOMER.CustomerId = ORDERS.CustomerId;
```

Session 11

Objective: Learning about different types of SQL Functions for Strings.

1 SUBSTRING()

The SUBSTRING() function extracts a substring from a string.
Syntax:

SUBSTRING(*string*, *start_pos*, *number_of_chars*)

Parameter	Description
string	Required. The string to extract from
start_pos	Required. The position to start extraction from. The first position in *string* is 1
number_of_chars	Required. The number of characters to extract

```
SELECT SUBSTRING('Quadratically',5);
SELECT SUBSTRING('Quadratically',-5,3);
SELECT SUBSTRING('foobarbar' FROM 4);
```

2 STRCMP()

The STRCMP() function tests whether two strings are the same.

Parameter	Description
string1 and string2	Required. The two strings to compare

- If *string1* and *string2* are the same, the STRCMP() function returns 0
- If *string1* is smaller than *string2*, the STRCMP() function returns -1
- If *string1* is large than *string2*, the STRCMP() function returns 1

```
SELECT STRCMP('mytesttext', 'mytesttext');
SELECT STRCMP('mytesttext', 'mytessttext');
```

3 LENGTH()

MySQL LENGTH() returns the length of a given string.

```
SELECT length(concat(Name,' ',Address)) as 'Length of Name with Address'
From customer;
```

4 UPPER()

This function coverts the contents of operand columns to uppercase letters.

```
SELECT upper(Name) as 'Name in Uppercase'
From customer;
```

4 LOWER()

This function coverts the contents of operand columns to lowercase letters.

```
SELECT lower(Name) as 'Name in Lowercase'
From customer;
```

5 CONCAT()

This function merges the contents of operands.

```
SELECT concat(Name,' ',Address) as 'Name with Address'
From customer;
```

Session 12

Objective: Learning about Subqueries.

1 SUBQUERIES

A subquery is a SQL query nested inside a larger query.

- A subquery may occur in a SELECT, FROM or WHERE CLAUSE.
- In MySQL subquery can be nested inside a SELECT, INSERT, UPDATE, DELETE, SET, or DO statement or inside another subquery.
- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.
- A subquery can be treated as an inner query, which a SQL query placed as a part of another query is called as outer query.
- The inner query executes first before its parent query so that the results of the inner query can be passed to the outer query.

2 Related Practices

```
SELECT *
FROM CUSTOMER
WHERE CustomerId IN (SELECT CustomerId
FROM CUSTOMER
WHERE Salary> 4500);

SELECT *
FROM CUSTOMER
WHERE CustomerId IN (SELECT CustomerId
FROM ORDERS
WHERE Amount > 500);

CREATE TABLE CUSTOMER_BKP
(
CustomerId int auto_increment PRIMARY KEY,
Name varchar(50) NOT NULL,
Age int NOT NULL CHECK (Age >= 18),
Address varchar(200) NULL  DEFAULT  'Dhaka',
```

```
Salary decimal(18,2) NULL
);

INSERT INTO CUSTOMER_BKP
SELECT Name,Age,Address,Salary FROM CUSTOMER
WHERE CustomerId IN (SELECT CustomerId
FROM CUSTOMER);

UPDATE CUSTOMER
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMER_BKP
WHERE AGE >= 27 );

SELECT Name,Age
FROM CUSTOMER
WHERE Salary >
(SELECT AVG(Salary) + 1000
FROM CUSTOMER );

SELECT * FROM EMPLOYEE
WHERE DepartmentId > ANY
(SELECT DepartmentId FROM DEPARTMENT WHERE Budget > 30000);

SELECT * FROM EMPLOYEE
WHERE DepartmentId > ALL
(SELECT DepartmentId FROM DEPARTMENT WHERE Budget > 30000);

SELECT EmployeeId,EmployeeName,DepartmentId
FROM EMPLOYEE
WHERE EXISTS
(SELECT Employee.DepartmentId FROM DEPARTMENT,EMPLOYEE
WHERE EMPLOYEE.DepartmentId = DEPARTMENT.DepartmentId
AND
DEPARTMENT.DepartmentId =6);
```

3 TASKS using SUBQUERIES

1. Find the name of the customers who are aged more than 23.

2. Find the name of the customers whose address starts with D.

3. Delete the orders less than of amount 200.

4. List the Name & Addresses of the persons who are aged more than the average age of the customers.

5. Find the date when maximum amount of order was placed.

6. Find the details of the employees who work in HR department.

7. Find the department name of the employee whose name starts with R.

8. Find the total number of employess in a department where the budget exceeds 50000.

9. Find the name & phone number of the employees if they work ina department which have budget greater than 350000.

10. Find the department names which have more then 2 sections.

11. Find the employes details of the department which have budget more than averged budget of the whole company.

12. Find the employes details of the department which have budget more than minimum budget of the whole company.

13. Find the department details which have atleast two working employees in it.

14. Find the department details which have employees living in Pabna.

15. Find the department details which have employees less than the average number of employees in whole company.

Session 13

Objective: Learning about Set Operations.

1 SQL UNION

The SQL **UNION operator** is used to combine the result sets of 2 or more SELECT statements. It removes duplicate rows between the various SELECT statements.

Each SELECT statement within the UNION must have the same number of fields in the result sets with similar data types, also same order, but they do not have to be the same length.

Basic Structure:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2
```

Example:

```
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA')
UNION
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA');
```

2 SQL UNION ALL

Same as UNION. The purpose of the SQL **UNION ALL** command is to combine the results of two queries together.

UNION and **UNION ALL** both combine the results of two SQL queries. The difference is that, while **UNION** only selects distinct values, **UNION ALL** selects all value

Basic Structure:

SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;

Example:

```
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA')
UNION ALL
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA');
```

3 SQL INTERSECT

The SQL **INTERSECT** clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements.

Just as with the UNION operator, the same rules apply when using the INTERSECT operator.

Basic Structure:

SELECT column_name(s)
FROM table
INTERSECT
SELECT column_name(s)
FROM table;

Example:

```
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA')
INTERSECT
```

```
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='KHULNA');
```

4 SQL EXCEPT

The EXCEPT command operates on two SQL statements. It takes all the results from the first SQL statement, and then subtract out the ones that are present in the second SQL statement to get the final answer. If the second SQL statement includes results not present in the first SQL statement, such results are ignored.

Each SELECT statement within the EXCEPT query must have the same number of fields in the result sets with similar data types.

Basic Structure:

```
SELECT column_name(s)
FROM table
MINUS
SELECT column_name(s)
FROM table;
```

Example:

```
(SELECT EmployeeName,City
FROM EMPLOYEE
)
MINUS
(SELECT EmployeeName,City
FROM EMPLOYEE
WHERE City !='DHAKA');
```

MID TERM EXAMINATION

There will be a mid-term examination. Contents covered upto 7th week will be the syllabus of the exam,

FINAL TERM EXAMINATION

There will be a one-hour written examination. Different types of questions will be included such as MCQ, mathematics, write SQL commands, find output etc.

PROJECT

For project submission students will have to work under a group. On the day of project submission individual viva will be taken for evaluation purpose.