Database management Systems
2018 Summer
Uttam Kumar Dey
Last Updated: May 21, 2018

Lab 1: Introduction to RDBMS and SQL

1. Background

This introductory lab is designed for students that are new to databases and have no or limited knowledge about databases and SQL. The instructions first introduce some basic relational databases concepts and SQL, and then show a number of tools that allow users to interact with MySQL databases. Lastly, the instructions point to a short and simple SQL tutorial using MySQL database and ask the students to write a few simple queries against this database.

2. Tutorial

2.1. What is SQL?

As it is stated in Wikipedia, SQL (Structured Query Language) is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS). SQL is a simple and unified language, which offers many features to make it as a powerfully diverse that people can work with a secure way. Using SQL language, one can retrieve data from a database, insert new records into databases, update/modify and delete existed records. SQL is also a ANSI standard computer language in the field of database, and supported by most commercial and free/open-source database systems, for instances the Oracle, DB2, Informix, SQL Server, MySQL, PostgreSQL, Access (Microsoft Office) and so on. Because of its simplicity, SQL is quite easy to learn and becoming the main interface for both users and developers to manipulate their data stored in relational database management systems (RDBMS).

The first version of SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. It was later formally standardized by the American National Standards Institute (ANSI) in 1986. Subsequent versions of the SQL standard have been released as International Organization for Standardization (ISO) standards (cited from Wikipedia). In the subsequent versions of SQL standards, many other extension have been adopted, such as procedural constructs, control-of-flow statements, user-defined data types, and various other language extensions. Among all the versions, the SQL: 1999 is the most remarkable one which formally accepts many of the above extension as part of the ISO standard for SQL, even the support for geographical data types (SQL99 MM/Spatial).

Although SQL is an ANSI and ISO standard computer language for accessing and manipulating data within database systems, different vendors might have proprietary extensions and supports for special but non-standard data types in their version of SQL. Anyway, in order to be in compliance with the ANSI/ISO standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

2.2. Basic Concepts in Relation Database Systems

In relational database systems, the data are organized as tables. In some cases, tables for the same purpose (e. g. tables used in the same application) could be grouped and saved in a database.

A data table (or simply a table) denotes the information needed to describe a certain kind/class of objects in the real world. Taking the information system of a university for the example, you can create tables for information about students (table 'Student'), schools (table 'School'), departments (table 'Department'), courses (table 'Course') and so on.

A table is made of one or more columns (or can be called as fields / attributes). Each field/column/attribute corresponds to a certain aspect of information used to describe an object. Taking the table of 'Student' for the example, you might need to store the name, gender, ID, date of birth, year of entering the school, the department and school she or he belongs to, and so on. Then for the students, you might have one following table in your database.

Table 1. Example of a Data Table in a Relational Database

ID	First Name	Last Name	Gender	Entering Year	School	Department

In order to create a table, you need to provide definitions for all the columns that make up this table. For one particular column, you need to specify its name (there is a certain naming rule for tables and columns in SQL), data type (such as Integer, String/Text, Date/Time, or even floating point numbers), and validation rules for a correct input. For example, the age is always recorded as an Integer, and for human beings, the possible age is within the range of 0 – 150 (this is called validation rule or constraint conditions). If a new input violates the constraints, the database will refuse to let it in and prompt the user an error report about this. A proper data type and constraint setting will help to ensure the quality of your data. A possible specification for the Table 1 might look as the follow. Here we are only talking about the constraints that involving a single column itself. More complex validation rules could be established on the table level involving several columns within the same table or columns from different tables. Such validation rules are also called reference integrity.

A row of data in the table (e.g. Table 1), which is also called a record, refers to a complete set of information for a concrete object in the real world. For example, each row of the 'Students' table represents all the information you need for a student in the system. In most cases, we need a unique ID or serial number for each record in the data table in order to easily and quickly identify the object you refer to in a transaction. Then you can set a single column or a collection of several columns, whose value or combination of values in different record is unique throughout the table, as the unique key (or called primary key) of this data table.

2.3. Fundamental Operations of SQL

SQL is the language you can use to express your request on the database systems. You could do various kinds of tasks on database systems using SQL, including the data creation, modification (insert/delete/update), information retrieval and database maintaining jobs. When using SQL, one only needs to express what she/he wants rather than how to guide the database to do it. For example, you might write a SQL command (or called a SQL statement) to find the highest student in the university, but you don't need to tell the database system to go through each record of the student table, pick up the height column and then compare the height of one student with that of another to find the largest height, and finally return the information of highest student to the client.

In this lab, we only provide some very simple example of SQL to give you a brief idea of what SQL looks like and how it works. Generally speaking, the SQL statements could be classified as three groups.

(1) The Data Manipulation Language (DML)

As the name tells, the DML is used to manipulate the data within database. One can send out data retrieval request using SELECT command, or modify the existing data stored in tables. The Data Manipulation Language (DML) part of SQL mainly includes:

- 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

(2) The Data Definition Language (DDL)

The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. You can also define indexes (keys), specify links between tables, and impose constraints between database tables. The most important 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

(3) The Data Control Language (DCL)

DCL is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

- CREATE ROLE/USER – create a user or role
- GRANT DELETE/UPDATE/CREATE – grant certain privilege to users or roles
- REVOKE DELETE/UPDATE/CREATE – revoke certain privilege from users or roles

Connecting to and Disconnecting from the Server

To connect to the server, you will usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you will also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user –p
 Enter password: ********
```

host and user represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The ******** represents your password; enter it when mysql displays the Enter password: prompt.

If that works, you should see some introductory information followed by a mysql> prompt:

```
shell> mysql -h host -u user –p
 Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
 Your MySQL connection id is 25338 to server version: 5.5.55-standard
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

The mysql> prompt tells you that mysql is ready for you to enter SQL statements.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell> mysql -u user -p
```

If, when you attempt to log in, you get an error message such as ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2), it means that the MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of Installing and Upgrading MySQL that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see Common Errors When Using MySQL Programs.

Some MySQL installations permit users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking mysql without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing QUIT (or \q) at the mysql> prompt:

```
mysql> QUIT
Bye
```

<u>Some MySQL Commands:</u>

1. Create the empty database with this command:

```
CREATE DATABASE;
```

2. To show all the databases:

```
SHOW DATABASES;
```

3. Start using a new database with the command:

```
USE dbname;
```

4. Showing all the tables created in a database:

```
SHOW TABLES;
```

5. View the configuration of a table:

```
DESCRIBE tablename;
```

6. Show all rows in a table:

```
SELECT * FROM tablename;
```

7. Removing an entire database:

```
DROP DATABASE dbname;
```

8. Removing an entire table from the database:

```
DROP TABLE tablename;
```

9. Removing all the rows/tuples from a table:

```
DELETE FROM tablename;
```

10. Show My SQL version and current date:

```
SELECT VERSION(), CURRENT_DATE;
```

11. Show user and current time:

```
   SELECT USER(), NOW();
```

Lab 2: Introduction to Relational Database and SQL

Task: In this lab, we are going to create a relational database for students: We will create tables and we will use alter statement to modify table structure

<u>Create table</u>

```
CREATE TABLE [IF NOT EXISTS] table_name(
    column definition
    ) engine=database engine name
```

To define a column for the table in the CREATE TABLE statement, you use the following syntax:
column_name data_type[size] [NOT NULL|NULL] [DEFAULT value] [AUTO_INCREMENT]

The most important components of the syntax above are:

- The column_name specifies the name of the column. Each column has a specific data type and the size e.g.,VARCHAR(255)
- The NOT NULL or NULL indicates that the column accepts NULL value or not.
- The DEFAULT value is used to specify the default value of the column.
- The AUTO_INCREMENT indicates that the value of the column is increased automatically whenever a new row is inserted into the table. Each table has one and only one AUTO_INCREMENT column.
- If you want to set particular columns of the table as the primary key, you use the following syntax: PRIMARY KEY (col1,col2,...)

Storage Engine:

You need to specify the storage engine for the table in the engine clause. You can use any storage engine such as InnoDB, MyISAM, HEAP, EXAMPLE, CSV, ARCHIVE, MERGE FEDERATED or NDBCLUSTER. If you don’t declare the storage engine explicitly, MySQL will use InnoDB by default. InnoDB became the default storage engine since MySQL version 5.5. The InnoDB table type brings many benefits of relational database management system such as ACID transaction, referential integrity, and crash recovery. In previous versions, MySQL used MyISAM as the default storage engine.

Example:

```
CREATE TABLE IF NOT EXISTS studentinfo (
  studentid INT(11) NOT NULL AUTO_INCREMENT,
  firstname VARCHAR(45) NOT NULL,
  batch  varchar(10),
 department   varchar(60),
  PRIMARY KEY (studentid)
) ENGINE=InnoDB
```

Cloning or Copying a Table

- LIKE
  Use CREATE TABLE ... LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:
  CREATE TABLE new_tbl LIKE orig_tbl;

- [AS] query_expression
  To create one table from another, add a SELECT statement at the end of the CREATE TABLE statement:
  CREATE TABLE new_tbl AS SELECT * FROM orig_tbl;

ALTER TABLE statement

The MySQL ALTER TABLE statement is used to add, modify, or drop/delete columns in a table. The MySQL ALTER TABLE statement is also used to rename a table.

Add column in table

Syntax:The syntax to add a column in a table in MySQL (using the ALTER TABLE statement) is:

```
ALTER TABLE table_name
ADD new_column_name      column_definition
[ FIRST | AFTER column_name ];
```

Here:

table_name:The name of the table to modify.

new_column_name:The name of the new column to add to the table.

column_definition:The datatype and definition of the column (NULL or NOT NULL, etc).

FIRST | AFTER column_name: Optional. It tells MySQL where in the table to create the column. If this parameter is not specified, the new column will be added to the end of the table.

Example:

- ALTER TABLE studentinfo ADD email varchar(100) NOT NULL ;
- ALTER TABLE studentinfo ADD phone int(15) AFTER department ;

Add multiple columns in table

Syntax:The syntax to add multiple columns in a table in MySQL (using the ALTER TABLE statement) is:

```
ALTER TABLE table_name
ADD new_column_name column_definition
[ FIRST | AFTER column_name ],
ADD new_column_name column_definition
  [ FIRST | AFTER column_name ],
 ...
;
```

table_name:The name of the table to modify.

new_column_name:The name of the new column to add to the table.

column_definition:The datatype and definition of the column (NULL or NOT NULL, etc).

FIRST | AFTER column_name Optional: It tells MySQL where in the table to create the column. If this parameter is not specified, the new column will be added to the end of the table.

Example

```
ALTER TABLE studentinfo
  ADD   lastname  VARCHAR(45),
    AFTER firstname,
  ADD pic varchar(35)
    AFTER department;
```

This ALTER TABLE example will add two columns to the contacts table - last_name and first_name.

The last_name field will be created as a varchar(40) NOT NULL column and will appear after the contact_id column in the table. The first_name column will be created as a varchar(35) NULL column and will appear after the last_name column in the table.

Modify column in table

Syntax

The syntax to modify a column in a table in MySQL (using the ALTER TABLE statement) is:

```
ALTER TABLE table_name
  MODIFY column_name column_definition
    [ FIRST | AFTER column_name ];
```

table_name:The name of the table to modify.

column_name:The name of the column to modify in the table.

column_definition:The modified datatype and definition of the column (NULL or NOT NULL, etc).

FIRST | AFTER column_name:Optional. It tells MySQL where in the table to position the column, if you wish to change its position.

Example

```
ALTER TABLE contacts
  MODIFY last_name varchar(50) ;
```

Modify Multiple columns in table

Syntax

The syntax to modify multiple columns in a table in MySQL (using the ALTER TABLE statement) is:

```
ALTER TABLE table_name
  MODIFY column_name column_definition
    [ FIRST | AFTER column_name ],
  MODIFY column_name column_definition
    [ FIRST | AFTER column_name ],
  ...
;
```

table_name:The name of the table to modify.

column_name:The name of the column to modify in the table.

column_definition:The modified datatype and definition of the column (NULL or NOT NULL, etc).

FIRST | AFTER column_name:Optional. It tells MySQL where in the table to position the column, if you wish to change its position.

<u>Drop column in table</u>

Syntax:

```
ALTER TABLE table_name   DROP COLUMN column_name;
```

table_name:The name of the table to modify.

column_name:The name of the column to delete from the table.

Example:

```
ALTER TABLE studentino
  DROP COLUMN  pic;
```

<u>Rename column in table</u>

Syntax:

```
ALTER TABLE table_name
  CHANGE COLUMN old_name new_name
    column_definition
    [ FIRST | AFTER column_name ]
```

Example:

```
ALTER TABLE studentinfo   CHANGE COLUMN firstname  fname     varchar(20) NOT NULL;
```

<u>Rename table</u>

Syntax:

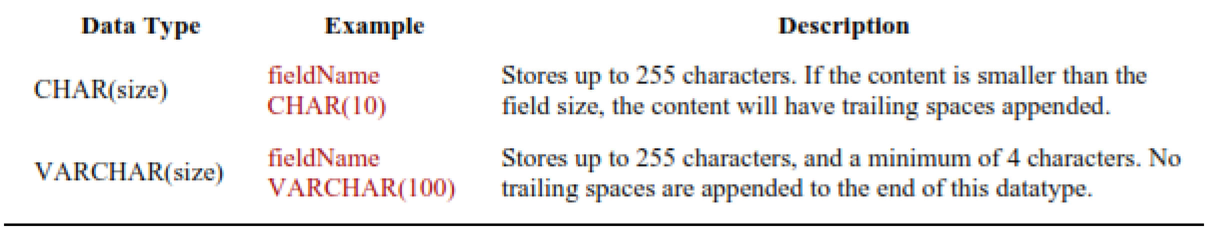
```
ALTER TABLE table_name   RENAME TO new_table_name;
```

Example

For example:

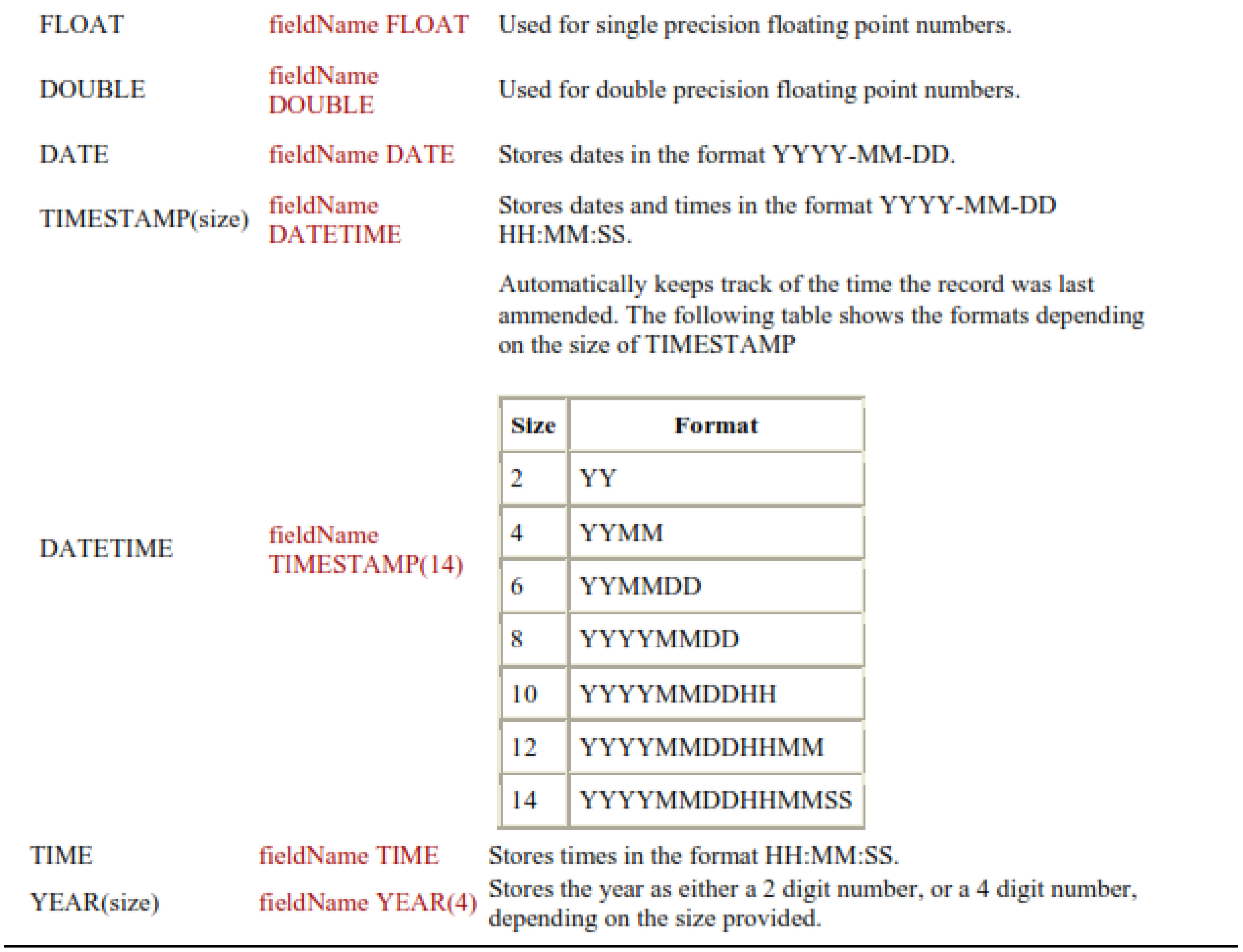
```
ALTER TABLE studentinfo   RENAME TO student;
```

<u>Data types in MySQL</u>



Data Type	Example	Description
CHAR(size)	fieldName CHAR(10)	Stores up to 255 characters. If the content is smaller than the field size, the content will have trailing spaces appended.
VARCHAR(size)	fieldName VARCHAR(100)	Stores up to 255 characters, and a minimum of 4 characters. No trailing spaces are appended to the end of this datatype.

		MySQL keeps track of a delimiter to keep track of the end of the field.
TINYTEXT	fieldName TINYTEXT	Stores up to 255 characters. Equivalent to VARCHAR(255).
TEXT	fieldName TEXT	Stores up to 65,535 characters. An Index can be created on the first 255 characters of a field with this data type.
MEDIUMTEXT	fieldName MEDIUMTEXT	Stores up to 16,777,215 characters. An Index can be created on the first 255 characters of a field with this data type.
LONGTEXT	fieldName LONGTEXT	Stores up to 4,294,967,295 characters. An Index can be created on the first 255 characters of a field with this data type. Note: The maximum size of a string in MySQL is currently 16 million bytes, so this data types is not useful at the moment.
ENUM	fieldName ENUM('Yes', 'No')	Stores up to 65,535 enumerated types. The DEFAULT modifier may be used to specify the default value for this field.
INT	fieldName INT	Stores a signed or unsigned integer number. Unsigned integers have a range of 0 to 4,294,967,295, and signed integers have a range of -2,147,438,648 to 2,147,438,647. By default, the INT data type is signed. To create an unsigned integer, use the UNSIGNED attribute. fieldName INT UNSIGNED The ZEROFILL attribute may be used to left-pad any of the integer with zero's. fieldName INT ZEROFILL The AUTO_INCREMENT attribute may be used with any of the Integer data types. The following example could be used to create a primary key using the AUTO_INCREMEMNT attribute. fieldName INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
TINYINT	fieldName TINYINT	Stores a signed or unsigned byte. Unsigned bytes have a range of 0 to 255, and signed bytes have a range of -128 to 127. By default, the TINYINT data type is signed.
MEDIUMINT	fieldName MEDIUMINT	Stores a signed or unsigned medium sized integer. Unsigned fields of this type have a range of 0 to 1,677,215, and signed fields of this type have a range of -8,388,608 to 8,388,607. By default, the MEDIUMINT data type is signed.
BIGINT	fieldName BIGINT	Stores a signed or unsigned big integer. Unsigned fields of this type have a range of 0 to 18,446,744,073,709,551,615, and signed fields of this type have a range of -9,223,372,036,854,775,808 to 9,223,327,036,854,775,807. By default, the BIGINT data type is signed.



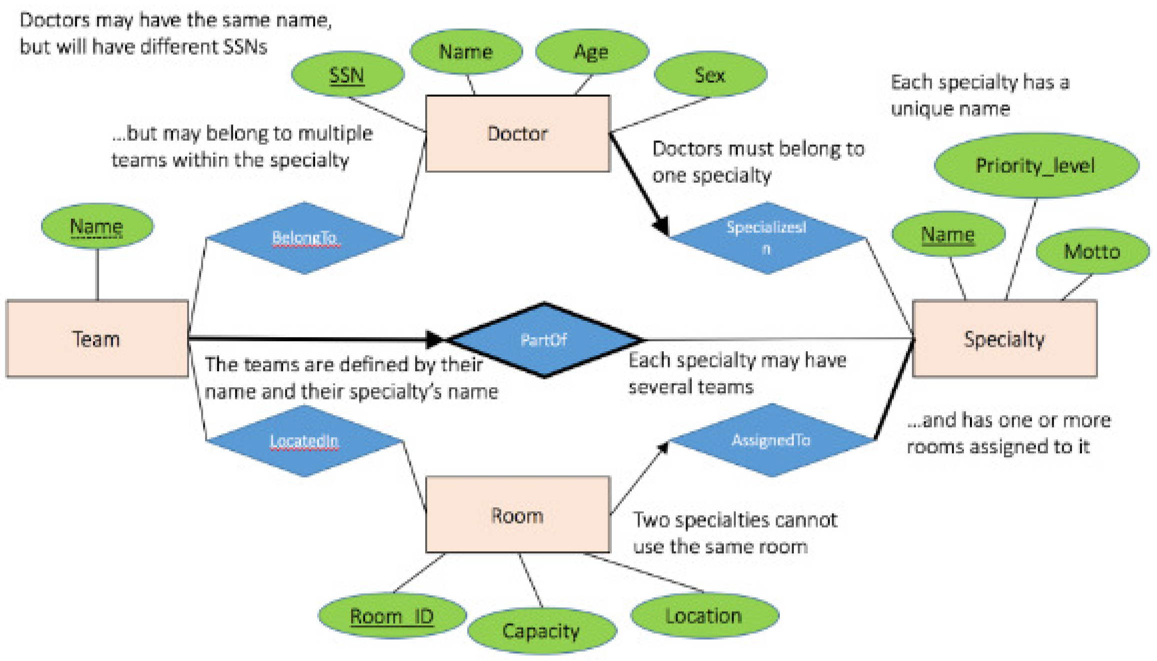
FLOAT	fieldName FLOAT	Used for single precision floating point numbers.
DOUBLE	fieldName DOUBLE	Used for double precision floating point numbers.
DATE	fieldName DATE	Stores dates in the format YYYY-MM-DD.
TIMESTAMP(size)	fieldName DATETIME	Stores dates and times in the format YYYY-MM-DD HH:MM:SS.
DATETIME	fieldName TIMESTAMP(14)	Automatically keeps track of the time the record was last ammended. The following table shows the formats depending on the size of TIMESTAMP
TIME	fieldName TIME	Stores times in the format HH:MM:SS.
YEAR(size)	fieldName YEAR(4)	Stores the year as either a 2 digit number, or a 4 digit number, depending on the size provided.

Size	Format
2	YY
4	YYMM
6	YYMMDD
8	YYYYMMDD
10	YYYYMMDDHH
12	YYYYMMDDHHMM
14	YYYYMMDDHHMMSS

Lab 3: Relational Database and Modeling

Design an ER diagram for a hospital that contains the following kinds of objects together with the listed attributes:

- Specialty(name, motto, priority level)
- Room(room ID, location, capacity)
- Team(name)
- Doctor(name, SSN, age, gender)


Doctors may have the same name, but will have different SSNs
SSN
Name
Age
Sex
Doctor
...but may belong to multiple teams within the specialty
Doctors must belong to one specialty
Each specialty has a unique name
Priority_level
Name
Motto
Name
BelongTo
Specializesin
Team
PartOf
Specialty
The teams are defined by their name and their specialty's name
Each specialty may have several teams
LocatedIn
AssignedTo
...and has one or more rooms assigned to it
Room
Two specialties cannot use the same room
Room_ID
Capacity
Location

Lab4: Select Statement examples

1. SELECT DISTINCT lastname FROM employees ORDER BY lastname;
   [When querying data from a table, you may get duplicate rows. In order to remove these duplicate rows, you use the `DISTINCT` clause in the SELECT statement.]

2. SELECT DISTINCT state, city FROM customers WHERE state IS NOT NULL ORDER BY state , city;
3. SELECT DISTINCT state FROM customers WHERE state IS NOT NULL LIMIT 5;
4. SELECT customername, country, creditLimit FROM customers WHERE (country = 'USA' OR country = 'France') AND creditlimit > 100000;
5. SELECT officeCode, city, phone, country FROM offices WHERE country IN ('USA' , 'France');
6. SELECT officeCode, city, phone FROM offices WHERE country NOT IN ('USA' , 'France');
7. SELECT productCode, productName, buyPrice FROM products WHERE buyPrice BETWEEN 90 AND 100;
8. SELECT productCode, productName, buyPrice FROM products WHERE buyPrice NOT BETWEEN 20 AND 100;

Updating table rows

The **UPDATE** command is used to modify data in a table. The syntax for this command is:

UPDATE
tablename
SET *columnname* = *expression* [, *columnname* = *expression*]
[WHERE *conditionlist*];

```
Example: UPDATE studentinfo
            SET   batch=38  WHERE sid=10001;
```

Deleting table rows

It is easy to delete a table row using the **DELETE** statement; the syntax is:

DELETE FROM *tablename*
    [WHERE *conditionlist*];

```
Eample: DELETE FROM  studentinfo WHERE sid=1005;
```