Assignment No.2

Title of Assignment: Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as Table, View, Index, Sequence, Synonym.

Objective: to learn and understand database creation, relation / table creation, setting index on table, sequences using MySQL DDL statements.

Relevant Theory

❖ SQL:

IBM developed the original version of SQL, originally called Sequel, as part of the System R project in the early 1970s. The Sequel language has evolved since then, and its name has changed to SQL (Structured Query Language). Many products now support the SQL language. SQL has clearly established itself as *the* standard relational database language. In 1986, the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) published an SQL standard, called SQL-86. ANSI published an extended standard for SQL, SQL-89, in 1989. The next version of the standard was SQL-92 standard, followed by SQL:1999, SQL:2003, SQL:2006, and most recently SQL:2008. The bibliographic notes provide references to these standards.
The SQL language has several parts:

- **Data-definition language** (DDL): The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.
- **Data-manipulation language** (DML): The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

❖ SQL Data Definition:

The set of relations in a database must be specified to the system by means of a data-definition language (DDL). The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:

- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk.

❖ Create Database:

You would need special privileges to create or to delete a MySQL database. So assuming you have access to root user, you can create any database using MySQL.
Example:
Here is a simple example to create database called **TUTORIALS**:

```
MySQL> create TUTORIALS
```

This will create a MySQL database TUTORIALS.

❖ Delete / Drop Database:

You would need special privileges to create or to delete a MySQL database. So assuming you have access to root user, you can create any database using MySQL .
Be careful while deleting any database because you will lose your all the data available in your database.
Here is an example to delete a database created:

```
MySQL > drop TUTORIALS
```

This will give you a warning and it will confirm if you really want to delete this database or not.

```
Dropping the database is potentially a very bad thing to do.
Any data stored in the database will be destroyed.

Do you really want to drop the 'TUTORIALS' database [y/N] y
Database "TUTORIALS" dropped
```

❖ Selecting Databases:

Once you get connection with MySQL server, it is required to select a particular database to work with. This is because there may be more than one database available with MySQL Server.
Selecting MySQL Database from Command Prompt:
This is very simple to select a particular database from mysql> prompt. You can use SQL command **use** to select a particular database.

Example:
Here is an example to select database called **TUTORIALS**:

```
mysql> use TUTORIALS;
Database changed
mysql>
```

Now, you have selected TUTORIALS database and all the subsequent operations will be performed on TUTORIALS database.
NOTE: All the database names, table names, table fields name are case sensitive. So you would have to use proper names while giving any SQL command.

❖ Data Types in MySQL:

Properly defining the fields in a table is important to the overall optimization of your database. You should use only the type and size of field you really need to use; don't define a field as 10 characters wide if you know you're only going to use 2 characters. These types of fields (or columns) are also referred to as data types, after the **type of data** you will be storing in those fields.
MySQL uses many different data types broken into three categories: numeric, date and time, and string types.

1. Numeric Data Types:

MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you. The following list shows the common numeric data types and their descriptions:

- **INT** - A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** - A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** - A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- **MEDIUMINT** - A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** - A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
- **FLOAT(M,D)** - A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
- **DOUBLE(M,D)** - A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
- **DECIMAL(M,D)** - An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

2. Date and Time Types:

The MySQL date and time datatypes are:

- **DATE:** A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31. For example, December 30th, 1973 would be stored as 1973-12-30.
- **DATETIME** - A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59. For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.
- **TIMESTAMP** - A timestamp between midnight, January 1, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).
- **TIME** - Stores the time in HH:MM:SS format.

- **YEAR(M):** Stores a year in 2-digit or 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be 1970 to 2069 (70 to 69). If the length is specified as 4, YEAR can be 1901 to 2155. The default length is 4.

3. String Types:
Although numeric and date types are fun, most data you'll store will be in string format. This list describes the common string datatypes in MySQL.

- **CHAR(M)** : A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
- **VARCHAR(M)** - A variable-length string between 1 and 255 characters in length; for example VARCHAR(25). You must define a length when creating a VARCHAR field.
- **BLOB or TEXT:** A field with a maximum length of 65535 characters. BLOBs are "Binary Large Objects" and are used to store large amounts of binary data, such as images or other types of files. Fields defined as TEXT also hold large amounts of data; the difference between the two is that sorts and comparisons on stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields. You do not specify a length with BLOB or TEXT.
- **TINYBLOB or TINYTEXT** - A BLOB or TEXT column with a maximum length of 255 characters. You do not specify a length with TINYBLOB or TINYTEXT.
- **MEDIUMBLOB or MEDIUMTEXT** - A BLOB or TEXT column with a maximum length of 16777215 characters. You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
- **LONGBLOB or LONGTEXT** - A BLOB or TEXT column with a maximum length of 4294967295 characters. You do not specify a length with LONGBLOB or LONGTEXT.
- **ENUM** - An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL). For example, if you wanted your field to contain "A" or "B" or "C", you would define your ENUM as ENUM ('A', 'B', 'C') and only those values (or NULL) could ever populate that field.

❖ Creating Table/Relation:

The table creation command requires:

- Name of the table
- Names of fields
- Definitions for each field

Syntax:
Here is generic SQL syntax to create a MySQL table:

```
CREATE TABLE table_name (column_name column_type);
```

Now, we will create following table in **TUTORIALS** database.

```
tutorials_tbl(
   tutorial_id INT NOT NULL AUTO_INCREMENT,
   tutorial_title VARCHAR(100) NOT NULL,
   tutorial_author VARCHAR(40) NOT NULL,
```

```
   submission_date DATE,
   PRIMARY KEY ( tutorial_id )
);
```

Here few items need explanation:

- Field Attribute **NOT NULL** is being used because we do not want this field to be NULL. So if user will try to create a record with 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 primary key. You can use multiple columns separated by comma to define a primary key.

Creating Tables from Command Prompt:
This is easy to create a MySQL table from mysql> prompt. You will use SQL command **CREATE TABLE** to create a table.
Example:
Here is an example, which creates **tutorials_tbl**:

```
mysql> use TUTORIALS;
Database changed
mysql> CREATE TABLE tutorials_tbl(
   -> tutorial_id INT NOT NULL AUTO_INCREMENT,
   -> tutorial_title VARCHAR(100) NOT NULL,
   -> tutorial_author VARCHAR(40) NOT NULL,
   -> submission_date DATE,
   -> PRIMARY KEY ( tutorial_id )
   -> );
Query OK, 0 rows affected (0.16 sec)
mysql>
```

NOTE: MySQL does not terminate a command until you give a semicolon (;) at the end of SQL command.

❖ Deleting / Dropping Table :

It is very easy to drop an existing MySQL table, but you need to be very careful while deleting any existing table because data lost will not be recovered after deleting a table.

Syntax:
Here is generic SQL syntax to drop a MySQL table:

```
DROP TABLE table_name ;
```

Dropping Tables from Command Prompt:
This needs just to execute **DROP TABLE** SQL command at mysql> prompt.
Example:
Here is an example, which deletes **tutorials_tbl**:

```
mysql> use TUTORIALS;
Database changed
mysql> DROP TABLE tutorials_tbl
```

```
Query OK, 0 rows affected (0.8 sec)
mysql>
```

❖ Inserting Data into Table:

To insert data into MySQL table, you would need to use SQL **INSERT INTO** command. You can insert data into MySQL table by using mysql> prompt or by using any script like PHP.

Syntax:

Here is generic SQL syntax of INSERT INTO command to insert data into MySQL table:

```
INSERT INTO table_name ( field1, field2,...fieldN )
                       VALUES
                       ( value1, value2,...valueN );
```

To insert string data types, it is required to keep all the values into double or single quote, for example:-"**value**".

Inserting Data from Command Prompt:

This will use SQL INSERT INTO command to insert data into MySQL table tutorials_tbl.

Example:

Following example will create 3 records into **tutorials_tbl** table:

```
mysql> use TUTORIALS;
Database changed
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
    ->VALUES
    ->("Learn PHP", "John Poul", NOW());
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
    ->VALUES
    ->("Learn MySQL", "Abdul S", NOW());
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
    ->VALUES
    ->("JAVA Tutorial", "Sanjay", '2007-05-06');
Query OK, 1 row affected (0.01 sec)
mysql>
```

NOTE: Please note that all the arrow signs (->) are not part of SQL command; they are indicating a new line and they are created automatically by MySQL prompt while pressing enter key without giving a semicolon at the end of each line of the command.

In the above example, we have not provided tutorial_id because at the time of table creation, we had given AUTO_INCREMENT option for this field. So MySQL takes care of inserting these IDs automatically. Here, **NOW()** is a MySQL function, which returns current date and time.

❖ Displaying Data from Table:

The SQL **SELECT** command is used to fetch data from MySQL database. You can use this command at mysql> prompt as well as in any script like PHP.
Syntax:
Here is generic SQL syntax of SELECT command to fetch data from MySQL table:

```
SELECT field1, field2,...fieldN table_name1, table_name2...
[WHERE Clause]
[OFFSET M ][LIMIT N]
```

- You can use one or more tables separated by comma to include various conditions using a WHERE clause, but WHERE clause is an optional part of SELECT command.
- You can fetch one or more fields in a single SELECT command.
- You can specify star (*) in place of fields. In this case, SELECT will return all the fields.
- You can specify any condition using WHERE clause.
- You can specify an offset using **OFFSET** from where SELECT will start returning records. By default offset is zero.
- You can limit the number of returns using **LIMIT** attribute.

Fetching Data from Command Prompt:
This will use SQL SELECT command to fetch data from MySQL table tutorials_tbl
Example:
Following example will return all the records from **tutorials_tbl** table:

```
mysql> use TUTORIALS;
Database changed
mysql> SELECT * from tutorials_tbl
+-------------+----------------+-----------------+-----------------+
| tutorial_id | tutorial_title | tutorial_author | submission_date |
+-------------+----------------+-----------------+-----------------+
|           1 | Learn PHP      | John Poul       | 2007-05-21      |
|           2 | Learn MySQL    | Abdul S         | 2007-05-21      |
|           3 | JAVA Tutorial  | Sanjay          | 2007-05-21      |
+-------------+----------------+-----------------+-----------------+
3 rows in set (0.01 sec)

mysql>
```

❖ Modifying Relation Schemas /Table Structure

MySQL **ALTER** command is very useful when you want to change a name of your table, any table field or if you want to add or delete an existing column in a table.
Let's begin with creation of a table called **testalter_tbl.**

```
root@host# mysql -u root -p password;
Enter password:*******
mysql> use TUTORIALS;
Database changed
mysql> create table testalter_tbl
    -> (
```

```
    -> i INT,
    -> c CHAR(1)
    -> );
Query OK, 0 rows affected (0.05 sec)
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
| c     | char(1) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Dropping, Adding or Repositioning a Column:
Suppose you want to drop an existing column **i** from above MySQL table then you will use **DROP** clause along with **ALTER** command as follows:

```
mysql> ALTER TABLE testalter_tbl  DROP i;
```

A **DROP** will not work if the column is the only one left in the table.
To add a column, use ADD and specify the column definition. The following statement restores the **i**column to testalter_tbl:

```
mysql> ALTER TABLE testalter_tbl ADD i INT;
```

After issuing this statement, testalter will contain the same two columns that it had when you first created the table, but will not have quite the same structure. That's because new columns are added to the end of the table by default. So even though **i** originally was the first column in mytbl, now it is the last one.

```
mysql> SHOW COLUMNS FROM testalter_tbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c     | char(1) | YES  |     | NULL    |       |
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

To indicate that you want a column at a specific position within the table, either use FIRST to make it the first column or AFTER col_name to indicate that the new column should be placed after col_name. Try the following ALTER TABLE statements, using SHOW COLUMNS after each one to see what effect each one has:

```
ALTER TABLE testalter_tbl DROP i;
ALTER TABLE testalter_tbl ADD i INT FIRST;
ALTER TABLE testalter_tbl DROP i;
```

```
ALTER TABLE testalter_tbl ADD i INT AFTER c;
```

The FIRST and AFTER specifiers work only with the ADD clause. This means that if you want to reposition an existing column within a table, you first must DROP it and then ADD it at the new position.

Renaming a Table:
To rename a table, use the **RENAME** option of the ALTER TABLE statement. Try out the following example to rename testalter_tbl to alter_tbl.

```
mysql> ALTER TABLE testalter_tbl RENAME TO alter_tbl;
```

❖ Database Constraint:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.
Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.
Following are commonly used constraints available in SQL.

- **NOT NULL Constraint:** Ensures that a column cannot have NULL value.
- **DEFAULT Constraint:** Provides a default value for a column when none is specified.
- **UNIQUE Constraint:** Ensures that all values in a column are different.
- **PRIMARY Key:** Uniquely identified each rows/records in a database table.
- **FOREIGN Key:** Uniquely identified a rows/records in any another database table.
- **CHECK Constraint:** The CHECK constraint ensures that all values in a column satisfy certain conditions.
- **INDEX:** Use to create and retrieve data from the database very quickly.

NOT NULL Constraint
By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such constraint on this column specifying that NULL is now not allowed for that column.
A NULL is not the same as no data, rather, it represents unknown data.
Example:
For example, the following SQL creates a new table called CUSTOMERS and adds five columns, three of which, ID and NAME and AGE, specify not to accept NULLs:

```
CREATE TABLE CUSTOMERS(
      ID   INT              NOT NULL,
      NAME VARCHAR (20)     NOT NULL,
      AGE  INT              NOT NULL,
      ADDRESS  CHAR (25) ,
      SALARY   DECIMAL (18, 2),
      PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to SALARY column in Oracle and MySQL, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
  MODIFY SALARY  DECIMAL (18, 2) NOT NULL;
```

DEFAULT Constraint:
The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.
Example:
For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, SALARY column is set to 5000.00 by default, so in case INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(
      ID   INT            NOT NULL,
      NAME VARCHAR (20)     NOT NULL,
      AGE  INT            NOT NULL,
      ADDRESS  CHAR (25) ,
      SALARY   DECIMAL (18, 2) DEFAULT 5000.00,
      PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a DFAULT constraint to SALARY column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
  MODIFY SALARY  DECIMAL (18, 2) DEFAULT 5000.00;
```

Drop Default Constraint:
To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE CUSTOMERS
  ALTER COLUMN SALARY DROP DEFAULT;
```

UNIQUE Constraint:
The UNIQUE Constraint prevents two records from having identical values in a particular column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having identical age.
Example:
For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, AGE column is set to UNIQUE, so that you can not have two records with same age:

```
CREATE TABLE CUSTOMERS(
      ID   INT            NOT NULL,
```

```
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL UNIQUE,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

If CUSTOMERS table has already been created, then to add a UNIQUE constraint to AGE column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
  MODIFY AGE INT NOT NULL UNIQUE;
```

You can also use following syntax, which supports naming the constraint in multiple columns as well:

```
ALTER TABLE CUSTOMERS
  ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);
```

DROP a UNIQUE Constraint:
To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE CUSTOMERS
  DROP CONSTRAINT myUniqueConstraint;
```

If you are using MySQL, then you can use the following syntax:

```
ALTER TABLE CUSTOMERS
  DROP INDEX myUniqueConstraint;
```

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.
A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a **composite key**.
If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).
Note: You would use these concepts while creating database tables.
Create Primary Key:
Here is the syntax to define ID attribute as a primary key in a CUSTOMERS table.

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
```

```
      SALARY   DECIMAL (18, 2),
      PRIMARY KEY (ID)
);
```

To create a PRIMARY KEY constraint on the "ID" column when CUSTOMERS table already exists, use the following SQL syntax:

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);
```

NOTE: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).
For defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE CUSTOMERS(
      ID   INT              NOT NULL,
      NAME VARCHAR (20)     NOT NULL,
      AGE  INT              NOT NULL,
      ADDRESS  CHAR (25) ,
      SALARY   DECIMAL (18, 2),
      PRIMARY KEY (ID, NAME)
);
```

To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax:

```
ALTER TABLE CUSTOMERS
  ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);
```

Delete Primary Key:
You can clear the primary key constraints from the table, Use Syntax:

```
ALTER TABLE CUSTOMERS DROP PRIMARY KEY ;
```

FOREIGN Key:
A foreign key is a key used to link two tables together. This is sometimes called a referencing key.
Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.
If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).
Example:
Consider the structure of the two tables as follows:
CUSTOMERS table:

```
CREATE TABLE CUSTOMERS(
      ID   INT              NOT NULL,
      NAME VARCHAR (20)     NOT NULL,
      AGE  INT              NOT NULL,
      ADDRESS  CHAR (25) ,
      SALARY   DECIMAL (18, 2),
      PRIMARY KEY (ID)
);
```

ORDERS table:

```
CREATE TABLE ORDERS (
      ID         INT       NOT NULL,
      DATE       DATETIME,
      CUSTOMER_ID INT references CUSTOMERS(ID),
      AMOUNT     double,
      PRIMARY KEY (ID)
);
```

If ORDERS table has already been created, and the foreign key has not yet been set, use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE ORDERS
  ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
```

DROP a FOREIGN KEY Constraint:
To drop a FOREIGN KEY constraint, use the following SQL:

```
ALTER TABLE ORDERS
  DROP FOREIGN KEY;
```

CHECK Constraint
The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered into the table.
Example:
For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you can not have any CUSTOMER below 18 years:

```
CREATE TABLE CUSTOMERS(
      ID   INT              NOT NULL,
      NAME VARCHAR (20)     NOT NULL,
      AGE  INT              NOT NULL CHECK (AGE >= 18),
      ADDRESS  CHAR (25) ,
      SALARY   DECIMAL (18, 2),
      PRIMARY KEY (ID)
```

```
);
```

If CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement similar to the following:

```
ALTER TABLE CUSTOMERS
   MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );
```

You can also use following syntax, which supports naming the constraint in multiple columns as well:

```
ALTER TABLE CUSTOMERS
   ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);
```

DROP a CHECK Constraint:
To drop a CHECK constraint, use the following SQL. This syntax does not work with MySQL:

```
ALTER TABLE CUSTOMERS
   DROP CONSTRAINT myCheckConstraint;
```

❖ Creating Index on Tables:

A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.
While creating index, it should be considered that what are columns which will be used to make SQL queries and create one or more indexes on those columns.
Practically, indexes are also type of tables, which keep primary key or index field and a pointer to each record into the actual table.
The users cannot see the indexes, they are just used to speed up queries and will be used by Database Search Engine to locate records very fast.
INSERT and UPDATE statements take more time on tables having indexes where as SELECT statements become fast on those tables. The reason is that while doing insert or update, database need to insert or update index values as well.

The INDEX is used to create and retrieve data from the database very quickly. Index can be created by using single or group of columns in a table. When index is created, it is assigned a ROWID for each row before it sorts out the data.
Proper indexes are good for performance in large databases, but you need to be careful while creating index. Selection of fields depends on what you are using in your SQL queries.
Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns:

```
CREATE TABLE CUSTOMERS(
      ID   INT              NOT NULL,
      NAME VARCHAR (20)     NOT NULL,
```

```
      AGE  INT            NOT NULL,
      ADDRESS  CHAR (25) ,
      SALARY   DECIMAL (18, 2),
      PRIMARY KEY (ID)
);
```

Now, you can create index on single or multiple columns using the following syntax:

```
CREATE INDEX index_name
   ON table_name ( column1, column2.....);
```

To create an INDEX on AGE column, to optimize the search on customers for a particular age, following is the SQL syntax:

```
CREATE INDEX idx_age
   ON CUSTOMERS ( AGE );
```

DROP an INDEX Constraint:
To drop an INDEX constraint, use the following SQL:

```
ALTER TABLE CUSTOMERS
  DROP INDEX idx_age;
```

❖ MySQL Sequence:

A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value and sequences provide an easy way to generate them. This chapter describes how to use sequences in MySQL.
Using AUTO_INCREMENT column:
The simplest way in MySQL to use Sequences is to define a column as AUTO_INCREMENT and leave rest of the things to MySQL to take care.
Example:
Try out the following example. This will create table and after that it will insert few rows in this table where it is not required to give record ID because it's auto incremented by MySQL.

```
mysql> CREATE TABLE insect
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    -> PRIMARY KEY (id),
    -> name VARCHAR(30) NOT NULL, # type of insect
    -> date DATE NOT NULL, # date collected
    -> origin VARCHAR(30) NOT NULL # where collected
);
Query OK, 0 rows affected (0.02 sec)
mysql> INSERT INTO insect (id,name,date,origin) VALUES
    -> (NULL,'housefly','2001-09-10','kitchen'),
    -> (NULL,'millipede','2001-09-10','driveway'),
    -> (NULL,'grasshopper','2001-09-10','front yard');
```

```
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM insect ORDER BY id;
+----+-------------+------------+------------+
| id | name        | date       | origin     |
+----+-------------+------------+------------+
|  1 | housefly    | 2001-09-10 | kitchen    |
|  2 | millipede   | 2001-09-10 | driveway   |
|  3 | grasshopper | 2001-09-10 | front yard |
+----+-------------+------------+------------+
3 rows in set (0.00 sec)
```

Renumbering an Existing Sequence:
There may be a case when you have deleted many records from a table and you want to resequence all the records. This can be done by using a simple trick but you should be very careful to do so if your table is having joins with other table.
If you determine that resequencing an AUTO_INCREMENT column is unavoidable, the way to do it is to drop the column from the table, then add it again. The following example shows how to renumber the id values in the insect table using this technique:

```
mysql> ALTER TABLE insect DROP id;
mysql> ALTER TABLE insect
    -> ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (id);
```

Starting a Sequence at a Particular Value:
By default, MySQL will start sequence from 1 but you can specify any other number as well at the time of table creation. Following is the example where MySQL will start sequence from 100.

```
mysql> CREATE TABLE insect
    -> (
    -> id INT UNSIGNED NOT NULL AUTO_INCREMENT = 100,
    -> PRIMARY KEY (id),
    -> name VARCHAR(30) NOT NULL, # type of insect
    -> date DATE NOT NULL, # date collected
    -> origin VARCHAR(30) NOT NULL # where collected
);
```

Alternatively, you can create the table and then set the initial sequence value with ALTER TABLE.

```
mysql> ALTER TABLE t AUTO_INCREMENT = 100;
```

❖ Creating Views

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depend on the written SQL query to create a view.
Views, which are kind of virtual tables, allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.
- Creating Views:

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view.
To create a view, a user must have the appropriate system privilege according to the specific implementation.
The basic CREATE VIEW syntax is as follows:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.
Example:
Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now, following is the example to create a view from CUSTOMERS table. This view would be used to have customer name and age from CUSTOMERS table:

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in similar way as you query an actual table. Following is the example:

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result:

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

The WITH CHECK OPTION:
The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.
If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.
The following is an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION:

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Dropping Views:
Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below:

```
DROP VIEW view_name;
```

Following is an example to drop CUSTOMERS_VIEW from CUSTOMERS table:

```
DROP VIEW CUSTOMERS_VIEW;
```

Conclusion:
Thus we have studied and understood SQL DDL statements with all database constrains successfully.