DBMS LEARN

Documentation

Abstract

In this module I learn basic concepts of DBMS and MySQL.

Dhruvil Dobariya
dhruvildobariya21@gmail.com

INDEX

INTRODUCTION TO DATABASE

1.1 INTRODUCTION

Data:

- Fact that ca be record or stored
- For ex: Person Name, Age, Gender and Weight…

Database:

- Collection of logically related data
- For ex: Books Database in Library, Student Database in University…

Management:

- Manipulation, Searching and Securing of data.
- Viewing result in GTU website, Searching exam papers in GTU website…

System:

- Program or tool that used to manage database
- MS SQL, MySQL, Postgres SQL, Oracle…

Database Management System:

- It is a software designed to define, manipulate, retrieve and manage data in a database.

1.2 ADVANTAGES

- Reduce data duplication
- Remove inconsistency
- Data isolation
- Guaranty of atomicity(0% or 100%)
- Allow implementing integrity constraints
- Sharing among the multiple user
- Restricted unauthorized access
- Provides backup and recovery services

1.3 BASIC TERMS

Data:

- Data is raw, unorganized facts that need to be processed.
- For ex: Marks of students…

Information:

- When data is processed, organized, structured or presented in a given context so as to make it useful, it is called information.
- For ex: Result of students (Pass or Fail)…

Metadata:

- Metadata is data about data.
- Data such as table name, column name, data type, authorized user and user access privileges for any table is called metadata for that table.

Data Dictionary:

- A data dictionary is an information repository which contains metadata.

Data Warehouse:

- A data warehouse is an information repository which stores data.

Field:

- A field is a character or group of characters that have a specific meaning.
- For ex: The value of Emp_Name, Address, Mobile_No etc are all fields of Faculty table.

Record/ Tuple:

- A record is a collection of logically related fields.
- For ex: The collection of fields (Emp_Name, Address, Mobile_No, Subject) forms a record for the Faculty.

Primary Key:

- A key which is unique as well as not null.

Unique Key:

- A key which is unique but it could be null.

Foreign Key:

- A key which liked two table.

Compose Key:

- A key that consists of multiple columns, because one column is not sufficiently identify record uniquely.

MySQL

2.1 Introduction

- MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses.
- MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company.
- MySQL is released under an open-source license.
- So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right.
- It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table.
- The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable.
- The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

OVERVIEW OF MYSQL WORKBENCH

3.1 INTRODUCTION

- MySQL Workbench is graphical user interface tool that used for working with database architects, developers, and Database Administrators.
- It is developed and maintained by Oracle.
- It provides SQL development, data modelling, data migration, and comprehensive administration tools for server configuration, user administration, backup, and many more.
- We can use this Server Administration for creating new physical data models, E-R diagrams, and for SQL development (run queries, etc.).
- It is available for all major operating systems like Mac OS, Windows, and Linux.
- MySQL Workbench fully supports MySQL Server version v5.6 and higher.

3.2 FUNCTIONALITY

SQL Development:

- This functionality provides the capability that enables you to execute SQL queries, create and manage connections to the database Servers with the help of built-in SQL editor.

Data Modelling (Design):

- This functionality provides the capability that enables you to create models of the database Schema graphically, performs reverse and forward engineering between a Schema and a live database, and edit all aspects of the database using the comprehensive Table editor.
- The Table editor gives the facilities for editing tables, columns, indexes, views, triggers, partitioning, etc.

Server Administration:

- This functionality enables you to administer MySQL Server instances by administering users, inspecting audit data, viewing database health, performing backup and recovery, and monitoring the performance of MySQL Server.

Data Migration:

- This functionality allows you to migrate from Microsoft SQL Server, SQLite, Microsoft Access, PostgreSQL, Sybase ASE, SQL Anywhere, and other RDBMS tables, objects, and data to MySQL.
- It also supports migrating from the previous versions of MySQL to the latest releases.

MySQL Enterprise Supports:

- This functionality gives the support for Enterprise products such as MySQL firewall, MySQL Enterprise Backup, and MySQL Audit.

3.3 EDITIONS

- MySQL Workbench is mainly available in three editions…
  - Community Edition (Open Source, GPL)
  - Standard Edition (Commercial)
  - Enterprise Edition (Commercial)

Community Edition:

- The Community Edition is an open-source and freely downloadable version of the most popular database management system.
- It came under the GPL license and is supported by a hug community of developers.

Standard Edition:

- It is the commercial edition that provides the capability to deliver high-performance and scalable Online Transaction Processing (OLTP) applications.
- It has made MySQL famous along with industrial-strength, performance, and reliability.

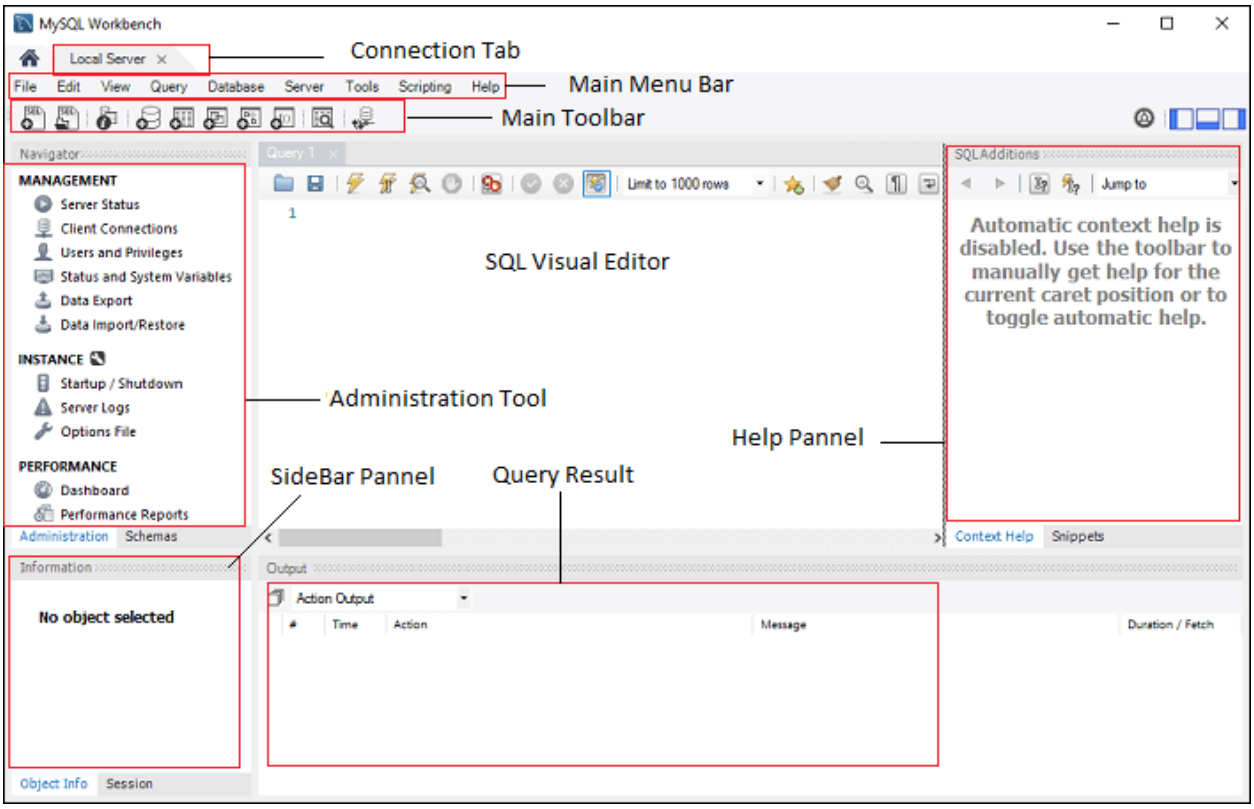
Enterprise Edition:

- It is the commercial edition that includes a set of advanced features, management tools, and technical support to achieve the highest scalability, security, reliability, and uptime.
- This edition also reduces the risk, cost, complexity in the development, deployment, and managing MySQL applications.

3.4 OVERVIEW

- When we open my sql workbench we have this type of window open.

DBMS Learn


MySQL Workbench
Local Server
Connection Tab
File Edit View Query Database Server Tools Scripting Help
Main Menu Bar
Main Toolbar
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Administration
Schemas
Administration Tool
Query 1
Limit to 1000 rows
SQL Visual Editor
SQLAdditions
Jump to
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.
Help Pannel
Context Help
Snippets
SideBar Pannel
Query Result
Information
No object selected
Object Info
Session
Output
Action Output
Time Action Message Duration / Fetch

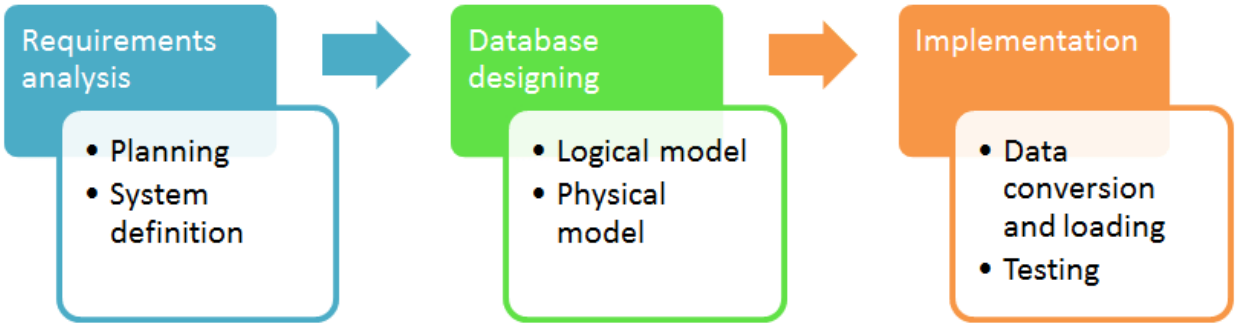
DATABASE DESIGN

4.1 INTRODUCTION

- Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems.
- Properly designed database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space.
- The database designer decides how the data elements correlate and what data must be stored.

4.2 DATABASE DEVELOPMENT LIFE CYCLE

- The database development life cycle has a number of stages that are followed when developing database systems.
- But it is not necessary to follow every stapes.



4.2.1 REQUIREMENT ANALYSIS:

Planning:

- This stages of database design concepts are concerned with planning of entire Database Development Life Cycle.
- It takes into consideration the Information Systems strategy of the organization.

System definition:

- This stage defines the scope and boundaries of the proposed database system.

4.2.2 DATABASE DESIGNING:

Logical model:

- This stage is concerned with developing a database model based on requirements.
- The entire design is on paper without any physical implementations or specific DBMS considerations.

Physical model:

- This stage implements the logical model of the database taking into account the DBMS and physical implementation factors.

4.2.3 Implementation:

Data conversion and loading:

- This stage of relational databases design is concerned with importing and converting data from the old system into the new database.

Testing:

- This stage is concerned with the identification of errors in the newly implemented system.
- It checks the database against requirement specifications.

4.3 Database Design Technique

- We have two types of database design techniques.
  - Normalization
  - ER Modeling

4.3.1 ER Modeling

- Entity Relationship Model (ER Modeling) is a graphical approach to database design.
- It is a high-level data model that defines data elements and their relationship for a specified software system.
- An ER model is used to represent real-world objects.

4.3.2 Normalization

- Normalization is the process of removing redundant data from tables to improve data integrity(completeness, accuracy and consistency of data), scalability and storage efficiency.
- We have 6 type of normal forms
  - 1NF
  - 2NF
  - 3NF
  - BCNF

- 4NF
- 5NF

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

BASIC OF SQL

5.1 DATATYPES

- We have four type of datatype
  - String
  - Numeric
  - Date/Time
  - Large Object

5.1.1 STRING

Data Type Syntax	Maximum Size	Description
CHAR(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Fixed-length strings. Space padded on right to equal size characters.
VARCHAR(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Variable-length string.
TINYTEXT(size)	Maximum size of 255 characters.	Where size is the number of characters to store.
TEXT(size)	Maximum size of 65,535 characters.	Where size is the number of characters to store.
MEDIUMTEXT(size)	Maximum size of 16,777,215 characters.	Where size is the number of characters to store.
LONGTEXT(size)	Maximum size of 4GB or 4,294,967,295 characters.	Where size is the number of characters to store.
BINARY(size)	Maximum size of 255 characters.	Where size is the number of binary characters to store. Fixed-length strings. Space padded on right to equal size characters. (Introduced in MySQL 4.1.2)
VARBINARY(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Variable-length string. (Introduced in MySQL 4.1.2)

5.1.2 NUMERIC DATATYPE

Data Type Syntax	Maximum Size	Description
BIT	Very small integer value that is equivalent to TINYINT(1). Signed values range from -128 to 127. Unsigned values range from 0 to 255.	

TINYINT(m)	Very small integer value. Signed values range from -128 to 127. Unsigned values range from 0 to 255.	
SMALLINT(m)	Small integer value. Signed values range from -32768 to 32767. Unsigned values range from 0 to 65535.	
MEDIUMINT(m)	Medium integer value. Signed values range from -8388608 to 8388607. Unsigned values range from 0 to 16777215.	
INT(m)	Standard integer value. Signed values range from -2147483648 to 2147483647. Unsigned values range from 0 to 4294967295.	
INTEGER(m)	Standard integer value. Signed values range from -2147483648 to 2147483647. Unsigned values range from 0 to 4294967295.	This is a synonym for the INT datatype.
BIGINT(m)	Big integer value. Signed values range from -9223372036854775808 to 9223372036854775807. Unsigned values range from 0 to 18446744073709551615.	
DECIMAL(m,d)	Unpacked fixed point number. m defaults to 10, if not specified. d defaults to 0, if not specified.	Where m is the total digits and d is the number of digits after the decimal.
DEC(m,d)	Unpacked fixed point number. m defaults to 10, if not specified. d defaults to 0, if not specified.	Where m is the total digits and d is the number of digits after the decimal. This is a synonym for the DECIMAL datatype.
NUMERIC(m,d)	Unpacked fixed-point number. m defaults to 10, if not specified. d defaults to 0, if not specified.	Where m is the total digits and d is the number of digits after the decimal. This is a synonym for the DECIMAL datatype.
FIXED(m,d)	Unpacked fixed-point number. m defaults to 10, if not specified. d defaults to 0, if not specified.	Where m is the total digits and d is the number of digits after the decimal. (Introduced in MySQL 4.1)

		This is a synonym for the DECIMAL datatype.
FLOAT(m,d)	Single precision floating point number.	Where m is the total digits and d is the number of digits after the decimal.
DOUBLE(m,d)	Double precision floating point number.	Where m is the total digits and d is the number of digits after the decimal.
DOUBLE PRECISION(m,d)	Double precision floating point number.	Where m is the total digits and d is the number of digits after the decimal. This is a synonym for the DOUBLE datatype.
REAL(m,d)	Double precision floating point number.	Where m is the total digits and d is the number of digits after the decimal. This is a synonym for the DOUBLE datatype.
FLOAT(p)	Floating point number.	Where p is the precision.
BOOL	Synonym for TINYINT(1)	Treated as a boolean data type where a value of 0 is considered to be FALSE and any other value is considered to be TRUE.
BOOLEAN	Synonym for TINYINT(1)	Treated as a boolean data type where a value of 0 is considered to be FALSE and any other value is considered to be TRUE.

5.1.3 Date and Time

Data Type Syntax	Maximum Size	Description
DATE	Values range from '1000-01-01' to '9999-12-31'.	Displayed as 'YYYY-MM-DD'.
DATETIME	Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.	Displayed as 'YYYY-MM-DD HH:MM:SS'.
TIMESTAMP(m)	Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.	Displayed as 'YYYY-MM-DD HH:MM:SS'.

TIME	Values range from '-838:59:59' to '838:59:59'.	Displayed as 'HH:MM:SS'.
YEAR[(2|4)]	Year value as 2 digits or 4 digits.	Default is 4 digits.

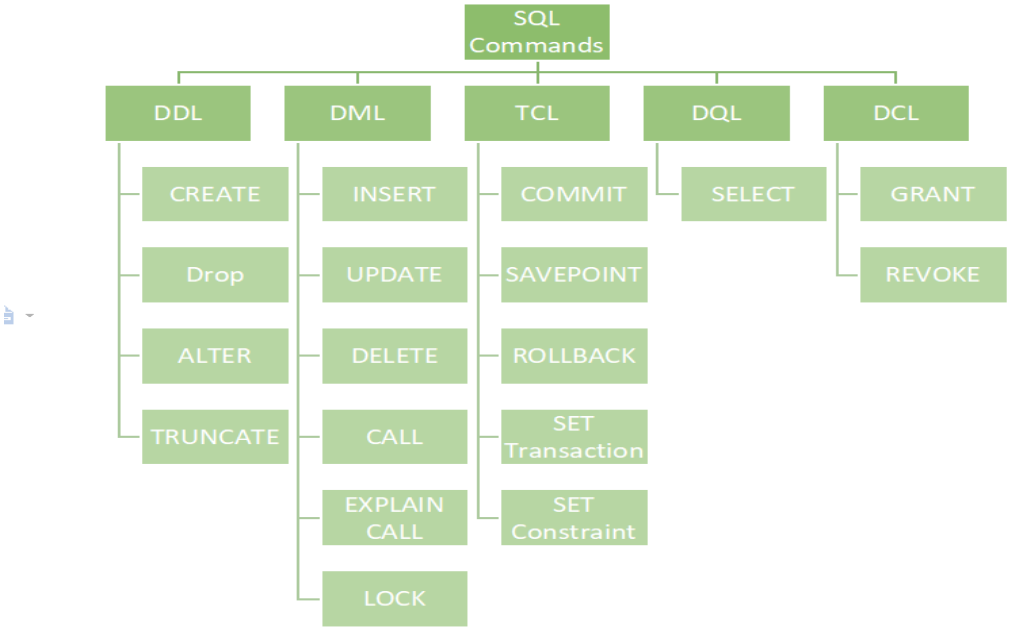
5.1.4 Large Object

Data Type Syntax	Maximum Size	Description
TINYBLOB	Maximum size of 255 bytes.	
BLOB(size)	Maximum size of 65,535 bytes.	Where size is the number of characters to store (size is optional and was introduced in MySQL 4.1)
MEDIUMBLOB	Maximum size of 16,777,215 bytes.	
LONGTEXT	Maximum size of 4GB or 4,294,967,295 characters.	

BASIC OPERATIONS

6.1 INTRODUCTION

- SQL have basic five components,
  - DDL - Data Definition Language
  - DQL - Data Query Language
  - DML - Data Manipulation Language
  - DCL - Data Control Language
  - TCL - Transaction Control Language



6.2 DATA DEFINITION LANGUAGE

- It contains SQL command that used for define schema.
- DDL is a set of SQL commands used to create, modify, and delete database structures but not data.
- DDL contains following commands,
  - Create
  - Drop
  - Alter

- Truncate
- Comment
- Rename

Create:

- This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).

Example:

```
Create Database CollageDB;

Create Table Student(
    Id int Not Null Auto_Increment,
    Name Varchar(250) Not Null,
    DateOfBirth Date Not Null,
    ContactNo Varchar(25),
    Gender Varchar(1),
    Primary Key(Id)
);
```

Drop:

- This command is used to delete objects from the database.

Example:

```
Drop Database CollageDB;

Drop Table Faculty;
```

Alter:

- This is used to update the structure of the database.

Example:

```
-- For Single Column
-- Add new column in table
Alter Table Faculty
Add Email Varchar(50);

-- Edit column in table
Alter Table Faculty
Modify Column Email Varchar(250);
```

```
-- Delete column in table
Alter Table Faculty
Drop Column Email;

-- For Multiple Column
-- Add new columns in table
Alter Table Faculty
Add Email Varchar(50),
Add Subject Varchar(50);

-- Edit columns in table
Alter Table Faculty
Modify Column Email Varchar(250),
Modify Column Subject Varchar (25);

-- Delete colomuns in table
Alter Table Faculty
Drop Column Email,
Drop Column Subject;
```

Truncate:

- This is used to remove all records from a table, including all spaces allocated for the records are removed.

Example:

```
Truncate Table student;
```

Comment:

- This is used to add comments to the data dictionary.

Example:

Rename:

- This is used to rename an object existing in the database.

Example:

```
-- Raname table name
Alter Table Faculty
Rename To FacultyNew
```

6.3 DATA QUERY LANGUAGE

- DQL is used to perform a query on schema.
- It is used to retrieve data from schema.
- It have only one command which is select.
- When we fired select command on table that time data stored in temporary table and this table we should see in output window.

Example:

```
-- Select all fields
Select * From Student;

-- select specific fields
Select Id, Name, Email From Student;

-- Where condition
Select * from Student
Where Id = 1;

Select * from Student
Where Id != 1;
-- OR
Where Id <> 1;

SELECT * from Student
Where RollNo > 5 and RollNo <= 10

SELECT * from Student
Where RollNo = 5 or RollNo = 10

SELECT * from Student
Where RollNo In(5, 10, 15, 20)

SELECT * from Student
Where RollNo BETWEEN 5 and 10
-- Between include uper bound and lower bound

SELECT * from Student
Where RollNo not BETWEEN 5 and 10

Select * from Student
Where Email is not Null;

-- Lilke
-- (_) represent one character
-- (%) represent more the one character

Select * from Student
Where RollNo like "1_";

Select * from Student
```

```
Where Name like "a%";
-- starting name from a

Select * from Student
Where Name like "%e";
-- starting name from e

Select * from Student
Where Name like "a%e";
-- starting name from a and ending from b

-- orderby
Select * from Student
ORDER BY Name, Email, RollNo, Id;

Select * from Student
ORDER BY Name Desc;

Select DISTINCT RollNo from Student

Select Name from Student
Limit 5;

Select Name as Username from Student;
```

6.4 Data Manipulation Language

- These commands is used for data manipulation in existing schema.
- It is the component of the SQL statement that controls access to data and to the database.
- Basically, DCL statements are grouped with DML statements.
- It contains following commands,
    - Insert
    - Update
    - Delete
    - Lock
    - Call
    - Explain Plan

Insert:

- It is used to insert data into a table.

Example:

```
-- Insert one record
Insert into Student (Name, DateOfBirth, ContactNo, Gender) values ("Dhruvil Dobariya", "2002-04-04", "9487587380", "M");
```

```
-- Insert multiple record
Insert into Student (Name, DateOfBirth, ContactNo, Gender) values
    ("Dhaval Dobariya", "2001-04-12","","M"),
    ("Bhargav Vachhani", "2002-01-04", "9408574858", ""),
    ("Jenil Vasoya", "2002-04-11", "", ""),
    ("Dhruv Rathod", "2002-07-11", "8594003858", "M");
```

Update:

- It is used to update existing data within a table.

Example:

```
Update Student
Set Name = "Dhruvi Savaliya", Gender = "F"
Where Id = 5
```

Delete:

- It is used to delete records from a database table.

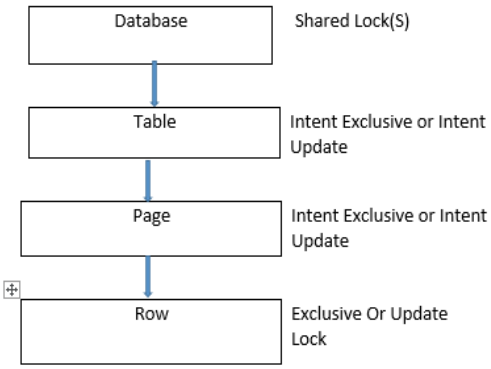
Example:

```
Delete From Student Where Id = 6
```

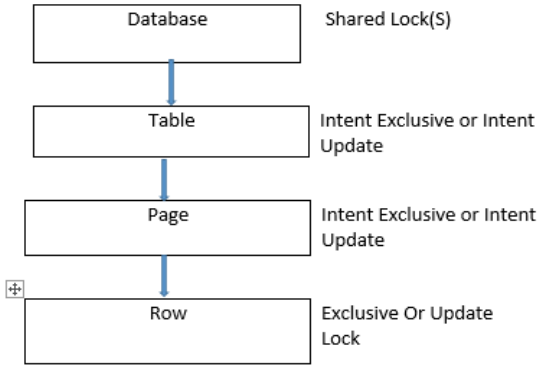
Lock:

- Data consistency is an important mechanism, and it can be done by means of SQL Locks.
- A lock is established in SQL Server when a transaction starts, and it will released when it is ended.
- We have different types of locks available in relational database,
    - **Shared (S) Locks:**
        - When the object needs to be read, this type of lock will occur.
        - But this is not harmful.
    - **Exclusive (X) Locks:**
        - It prevents other transactions like inserting/updating/deleting.
        - So no modifications can be done when we apply this type of lock on object.
    - **Update (U) Locks:**
        - It's like Exclusive lock but here the operation can be viewed as "read phase" and "write phase".
        - During the read phase, other transactions are prevented.
    - **Intent Locks:**

        - Intent lock happens on a table, when the shared (S) lock or exclusive (X) lock or Update (U) lock happens on the row.
    - **Regular intent locks:**
        - Intent exclusive (IX)
        - Intent shared (IS)
        - Intent update (IU).
    - **Conversion locks:**
        - Shared with intent exclusive (SIX)
        - Shared with intent update (SIU)
        - Update with intent exclusive (UIX)
- We have hierarchy for lock.



(Select)



(Update/Insert/Delete)

Call:

- Call a PL/SQL

Explain Plan:

- It describes the access path to data.

6.5 Data Control Language

- DCL includes commands which mainly use for user rights, permissions and other controls on database.
- It contains two command,
  - Grant
  - Revoke

Grant:

- This command is used to give user access privileges of database to user.

Example:

```
GRANT insert,
select on studentdb to root
-- We give permision of insrt into studentdb to root
```

Revoke:

- This command revoke the user privileges of database from the user.

Example:

```
REVOKE insert,
select on studentdb from root
-- We revoke permision of insrt into studentdb from root
```

6.6 Transaction Control Language

- We have group of some transection which used for execute single query.
- Transection done when this group of transections id done,
- If any one is failed then whole transection is failed.
- So transection have only two result, success and failure.
- Transection contains some commands,

- Begin
- Commit
- Rollback
- Savepoint
- Set Transection

Begin:

- Opens a Transaction.

Commit:

- Commits a Transaction.

Rollback:

- Rollback transection if any error occur during transaction.

Savepoint:

- Set a save point within the transection.

Set Transection:

- Specify characteristics for transection.

DATA SORTING

7.1 INTRODUCTION

- We have “Order By” key word to sort our result set.
- By default it’s sort in ascending order, But we can specify if we want to sort in descending using “Desc” Key word.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example:

```
-- ascending order
SELECT * FROM Customers
ORDER BY Country;

-- descending order
SELECT * FROM Customers
ORDER BY Country; DESC
```

NULL VALUE & KEYWORD

8.1 INTRODUCTION

- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field.
- Then, that field will be saved with a NULL value.
- Null value different from zero or empty.
- Null means nothing.
- We have two key work to check null value, “Is Null” and “Is Not Null”.

Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL | IS NOT NULL;
```

Example:

```
-- Get rows which have address is null
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;

-- Get rows which have address is not null
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

KEYS AND AUTO INCREMENT

9.1 PRIMARY KEY

- Primary key is key that used to uniquely identify record in table.
- Primary key must be unique and not null.
- One table contains one, primary key, but this primary key may combination one or more column.
- We are use “Primary Key” key word to define primary key.

```
-- Define Primary Key
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);

-- OR

CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);

-- Alter Primary Key
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
-- OR
ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);

-- Drop Primary Key
ALTER TABLE Persons
DROP PRIMARY KEY;
```

9.2 AUTO INCREMENT

- Auto Increment generate automatic unique and incremental number in particular field.

Example:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

9.3 Foreign Key

- Foreign key is the key that used to linked two table.
- Parent table primary key is used as a foreign key in child table.
- We have “Foreign Key” keyword to define foreign key.

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

9.4 Unique Key

- Unique key is the key that used to set unique behavior of particular field.
- Unique key may null, primary key must not.
- Unique key may one or more in table.
- We have “Unique Key” keyword to define unique key.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
-- OR
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
```

```
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

AGGREGATE FUNCTIONS

10.1 INTRODUCTION

- Aggregate function is used to perform calculation on row of single column.
- It return only single value.
- It is also used to summarize the data.

10.2 AGGREGATE FUNCTION

- We have five types of aggregate function,
  - Count
  - Sum
  - Avg
  - Min
  - Max

Count:

- Count number is used to count number of rows in table.

Example:

```
select Count(*) from Product;
select count(distinct Company) from Product;
```

Sum:

- Sum is used to calculate sum of all selected column.
- It works on only numeric fields.

Example:

```
SELECT Sum(Quantity) As TotalQuantity from Product;
```

Avg:

- Avg function is used to calculate average of selected column.
- It works on only numeric fields.

Example:

```
SELECT AVG(Cost) from Product;
```

Min:

- Min is used to find minimum value of particular column.
- It works on only numeric fields.

Example:

```
SELECT Min(Quantity) from Product;
```

Max:

- Max is used to find maximum value of particular column.
- It works on only numeric fields.

Example:

```
SELECT Max(Quantity) from Product;
```

10.3 Group By

- Group By is used to make collection of same value so we can summarize data.
- Group By statement is used with aggregate functions.

Example:

```
SELECT Company, Sum(Quantity) from Product
Group By Company
```

10.4 Having

- Having is used to specify condition after group by with aggregate function.
- We must use “Having” with aggregate function we can’t use “Where”.

Example:

```
SELECT Company, Count(Company) From Product
Group By Company
Having Count(Company) >= 5;

SELECT Company, Sum(Quantity) As TotalQuantity from Product
Group By Company
HAVING TotalQuantity > 50;
```

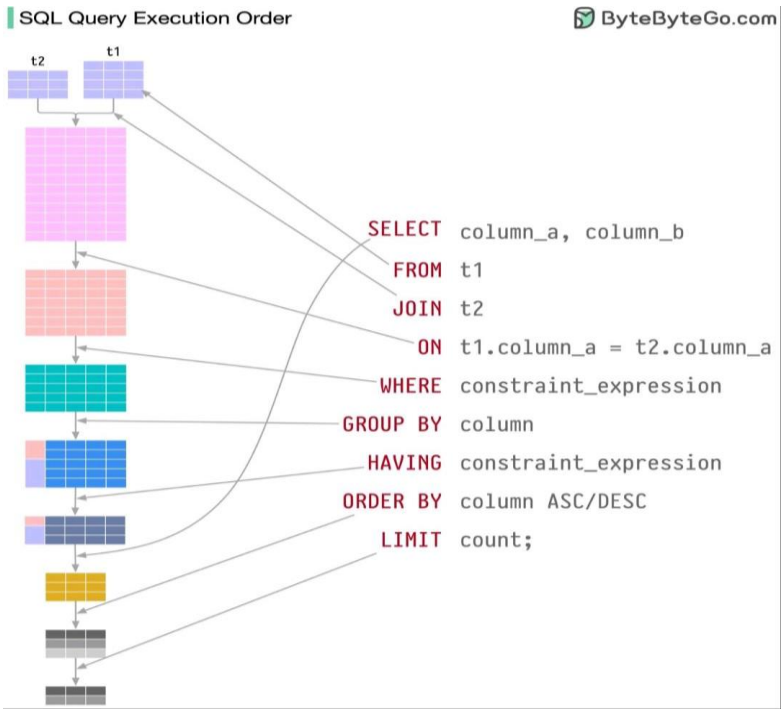
```sql
SELECT Company, Sum(Quantity) As TotalQuantity from Product
Where Quantity >= 4
Group By Company
HAVING TotalQuantity > 50;

SELECT Company, Sum(Quantity) As TotalQuantity from Product
Where Quantity >= 4
Group By Company
HAVING TotalQuantity > 40
ORDER BY Company
LIMIT 2;
```

10.5 SEQUENCE OF STATEMENT

- We have particular sequence that we must follow in SQL queries.

Syntax:

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



SUB QUERY

11.1 INTRODUCTION

- Sub query means query within the query or nesting of query.
- The outer query is called as main query and inner query is called as subquery.
- We can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause.
- Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator.
- It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
- The subquery generally executes first when the subquery doesn't have any co-relation with the main query.
- Subquery must be enclosed in parentheses.
- Subqueries are on the right side of the comparison operator.
- ORDER BY command cannot be used in a Subquery.
- GROUPBY command can be used to perform same function as ORDER BY command.

Example:

```
-- Example 1:
SELECT * From Student
Where RollNo IN
(
    SELECT RollNo FROM Result
    WHERE  Grade = "A"
);
-- Here we have two table and We want to get those student who beloges to the grade A.
-- Student Table{ Id, RollNo, Name, ContactNo}
-- Result Table{ Id, RollNo, Grade}

-- Example 2:
-- Given:
-- We have Two Division Table
-- 1) DivisionBCX{Id, RollNo, Name}
-- 2) DivisionBCW{Id, RollNo, Name}
-- Problem: We want to put one student from DivisionDCW to DivisionBCX which RollNo have 102.
INSERT into DivisionBCX(
    SELECT * FROM DivisionBCW
    Where RollNo = 102
);

-- Example 3:
-- Given:
-- We have Two Table
```

```
-- 1) City{CityId, City, StateId}
-- 2) State{StateId, State}
-- Problem: When we delete Gujarat then it's automatic delete all the city of Gujarat State
DELETE From City
Where StateId IN(
    Select StateId From State
    Where State = "Gujarat"
);
DELETE FROM State
Where State = "Gujarat";

-- Example 4:
-- Given:
-- We have Two Table
-- 1) Student Table{ Id, RollNo, Name, Division}
-- 2) Result Table{ Id, RollNo, Grade}
-- Problem: We want to promote student from their division to BCX division which student get A
grade.
UPDATE
    SET Division = "BCX"
Where RollNo IN(
    SELECT RollNo From Result
    Where Grade = "A"
);
```

11.2 EXISTS

- The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records.

Example:

```
-- We have two table
-- Product{ProductID, ProductName, SupplierID, CategoryID, Unit, Price}
-- Suplier{SupplierID, SupplierName, ContactName, Address, City, PostalCode, Country}
--  Problem: We ant to those supplier name that deliver product which price is more then 20
SELECT SupplierName FROM Suppliers
WHERE EXISTS (
    SELECT * FROM Products
    WHERE Products.SupplierID = Suppliers.SupplierID AND Price > 20
);

-- using In
SELECT SupplierName FROM Suppliers
WHERE SupplierID In (
    SELECT SupplierID FROM Products
    WHERE Price > 20
);
```

11.3 ANY AND ALL

- The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

11.3.1 ANY:

- Returns TRUE if ANY of the subquery values meet the condition
- ANY means that the condition will be true if the operation is true for any of the values in the range.

Example:

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY(
    SELECT ProductID
    FROM OrderDetails
    WHERE Quantity = 10
);
-- The following SQL statement lists the ProductName if it finds ANY records in the
OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column
has some values of 10)
```

11.3.2 ALL:

- Returns TRUE if ALL of the subquery values meet the condition
- It is used with SELECT, WHERE and HAVING statements
- ALL means that the condition will be true only if the operation is true for all values in the range.

Example:

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL(
    SELECT ProductID
    FROM OrderDetails
    WHERE Quantity = 10
);
-- The following SQL statement lists the ProductName if ALL the records in the OrderDetails
table has Quantity equal to 10. This will of course return FALSE because the Quantity column
has many different values (not only the value of 10):
```

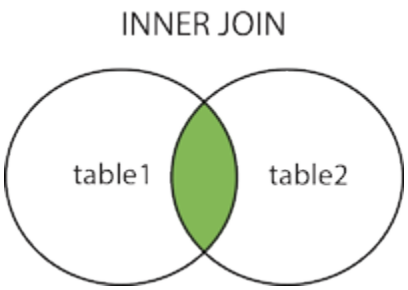
JOIN

12.1 INTRODUCTION

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- It is not compulsory that both reference column have same column name, but must have same datatype.
- We have two type of join in SQL.
    - Inner
    - Outer
        - Left
        - Right
        - Full

12.2 INNER JOIN

- In inner join we get table which contains row that should have the same value of reference column in both table.



Syntax:

```
Select * from tablename1
Inner Join tablename2 ON
tablename2.refrencecolumn = tablename1.refrencecolumn;
```

Example:

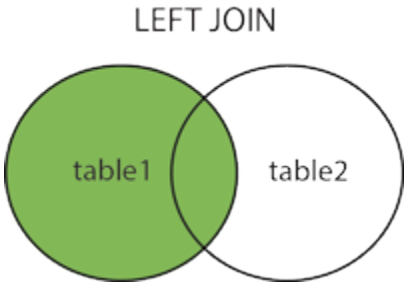
```
-- Inner Join
Select * from Student
Inner Join Collage ON
Collage.CollageId = Student.CollageId;
```

12.3 Outer Join

- In outer join we have three different join.
  - Left Join
  - Right Join
  - Full Join

12.3.1 Left Join

- In inner join we get table which contains all row of left table and those and from right table only match value of reference column.
- If any column not match value of reference column with right side table then these column value comes null.



Syntax:

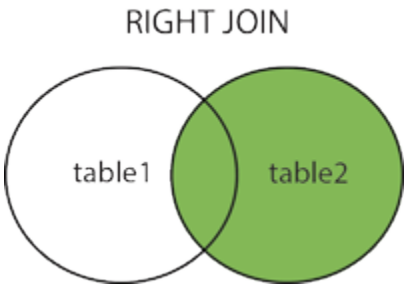
```
Select * from tablename1
Left Join tablename2 ON
tablename2.refrencecolumn = tablename1.refrencecolumn;
```

Example:

```
-- Inner Join
Select * from Student
Left Join Collage ON
Collage.CollageId = Student.CollageId;
```

12.3.2 Right Join

- In inner join we get table which contains all row of Right table and those and from left table only match value of reference column.
- If any column not match value of reference column with left side table then these column value comes null.



Syntax:

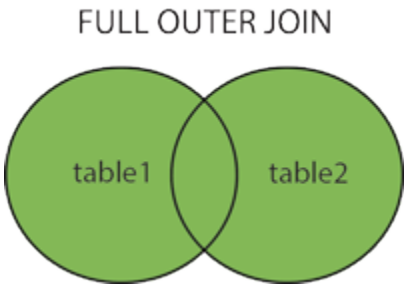
```
Select * from tablename1
Right Join tablename2 ON
tablename2.refrencecolumn = tablename1.refrencecolumn;
```

Example:

```
-- Inner Join
Select * from Student
Right Join Collage ON
Collage.CollageId = Student.CollageId;
```

12.3.3 FULL JOIN

- In inner join we get table which contains all row of right table and left table, both match value of reference column and not.
- If any column not match value of reference column then these column value comes null.



Syntax:

```
Select * from tablename1
Full Join tablename2 ON
tablename2.refrencecolumn = tablename1.refrencecolumn;
```

Example:

```
-- Inner Join
```

```
Select * from Student
Full Join Collage ON
Collage.CollageId = Student.CollageId;
```

UNION

13.1 INTRODUCTION

- It is used to combine result set of two select statements.
- Every result set must have same number of columns and same datatype sequence of column.
- We have two types of statement in union.
  - Union
  - Union All

13.2 UNION

- It union two result set and give distinct value in result set.

Syntax:

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

Example:

```
-- Union
Select City From Student
Union
Select City From Faculty;
```

13.3 UNION ALL

- It union two result set and I does not give distinct value in result set.

Syntax:

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

Example:

```
-- Union All
Select City From Student
Union All
Select City From Faculty;
```

INDEX

14.1 INTRODUCTION

- Indexes are used to retrieve data from the database more quickly.
- We use index on these column which frequently use to retrieve the data.
- Index columns that are used for joins to improve join performance.
- Avoid columns tat contains too many null.
- Small table do not required indexes.
- Primary key and Unique Key is automatically create index.

Syntax:

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

Example:

```
-- single index
Create Index EmailIndex
on indexlearn.user(Email);

-- multiple index
Create Index EmailGenderIndex
on indexlearn.user(Email, Gender);

-- Show Index
Show Index from indexlearn.user;

-- Drop Index
Alter Table User
Drop Index EmailIndex;
```

VIEW

15.1 INTRODUCTION

- View is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table.
- The fields in a view are fields from one or more real tables in the database.
- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
- It only allow select statement.

Stored Procedure	Function	View
Accepts Parameters	Accepts Parameters	Does NOT Accept Parameters
Can contain several statements	Can contain several statements	Can contain only one single SELECT query
Can call functions and views	Cannot call stored procedures but can call views	Cannot call stored procedures but can call functions
Can return multiple values/tables	Can return a single value/table	Can return a single table
Exceptions can be handled using try-catch block	Exceptions cannot be handled	Exceptions cannot be handled
Allows insert/update/delete/select	Only allows select statement	
CANNOT be used in a SELECT query	CAN be used in a SELECT query	CAN be used in a SELECT query
NOT mandatory to return a value	SHOULD return a value	SHOULD return a table

Syntax:

```
CREATE VIEW viewname
AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```
-- Create View
Create View StudentView
```

```
AS
Select StudentId, FirstName, LastName, Name from Student
Inner Join Collage ON
Collage.CollageId = Student.CollageId;

-- Alter View
Alter View StudentView
As
Select StudentId, FirstName, LastName, Name as CollageName from Student
Inner Join Collage ON
Collage.CollageId = Student.CollageId;

-- Rename Table StudentView
to StudentWithCollageNameView

-- Select View
Select * From StudentWithCollageNameView

-- Drop View
Drop View StudentWithCollageNameView
```

FUNCTIONS

- A stored function in MySQL is a set of SQL statements that perform some task/operation and return a single value.
- It is one of the types of stored programs in MySQL.
- Generally, we used this function to encapsulate the common business rules or formulas reusable in stored programs or SQL statements.
- The stored function is almost similar to the procedure in MySQL, but it has some differences that are as follows:
    - The function parameter may contain only the IN parameter but can't allow specifying this parameter, while the procedure can allow IN, OUT, INOUT parameters.
    - The stored function can return only a single value defined in the function header.
    - The stored function may also be called within SQL statements.
    - It may not produce a result set.
    - Thus, we will consider the stored function when our program's purpose is to compute and return a single value only or create a user-defined function.

Syntax:

```
Delimiter //
Create Function NameOfFunction
(
    parm1 typeOfparam1,...
)
Returns ReturnType
Begin
    -- sql statements
End //
Delimiter ;
```

Example:

```
Delimiter //
Create Function Addition
(
    a int,
 b int
)
Returns int
Begin
    Declare total int default 0;
    Set total = a + b;
    Return total;
```

```
End //
Delimiter ;

-- Select Function
Select Addition(2,3);

-- Show All functions
Show Function Status
Where db = "FunctionLearn";

-- Drop Function
Drop Function Addition;
```

16.1 How To Update Function?

```
ALTER FUNCTION func_name [characteristic ...]

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}
```

- This statement can be used to change the characteristics of a stored procedure.
- More than one change may be specified in an ALTER PROCEDURE statement.
- However, you cannot change the parameters or body of a stored procedure using this statement.
- To make such changes, you must drop and re-create the procedure using DROP PROCEDURE and CREATE PROCEDURE.

STORED PROCEDURE

17.1 INTRODUCTION

- A stored procedure is a prepared SQL code that we can save, so the code can be reused over and over again.
- So if we have an SQL query that we write over and over again, save it as a stored procedure, and then just call it to execute it.
- we can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
- Stored Procedure have three different types of parameters like,
    - In
    - Out
    - InOut

IN parameter

- It is the default mode.
- It takes a parameter as input, such as an attribute.
- When we define it, the calling program has to pass an argument to the stored procedure.
- This parameter's value is always protected.

OUT parameters:

- It is used to pass a parameter as output.
- Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program.
- It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

INOUT parameters:

- It is a combination of IN and OUT parameters.
- It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

Syntax:

```
DELIMITER //
Create Procedure ProcedureName
(
    In/Out/InOut param1 typeOfparam1,...,
```

```
)
Begin
    -- Sql statements
End //
DELIMITER;
```

Example:

```
DELIMITER //
Create Procedure GetStudentById
(
    In StudentId int
)
Begin
    Select
        Student.StudentId,
        Student.FirstName,
        Student.LastName,
        Student.Email,
        Student.Gender,
        Collage.Name As CollageName,
        City.CityName
    From
        Student
        Join Collage
        on Collage.CollageId = Student.CollageId
        Join City
        on City.CityId = Student.CityId
    Where
        Student.StudentId = StudentId;
End //
DELIMITER;

-- Call procedure
Call GetStudentById(1);

-- Show all procedure
Show Procedure Status
Where db = "JoinLearn";

-- Drop procedure
Drop PROCEDURE DelateStudent;
```

17.2 How to Update Stored Procedure?

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

```
}
```

- This statement can be used to change the characteristics of a stored procedure.
- More than one change may be specified in an ALTER PROCEDURE statement.
- However, we cannot change the parameters or body of a stored procedure using this statement.
- To make such changes, you must drop and re-create the procedure using DROP PROCEDURE and CREATE PROCEDURE.

TRIGGER

18.1 INTRODUCTION

- A trigger in MySQL is a set of SQL statements that reside in a system catalog.
- It is a special type of stored procedure that is invoked automatically in response to an event.
- Each trigger is associated with a table, which is activated on any DML statement such as INSERT, UPDATE, or DELETE.
- A trigger is called a special procedure because it cannot be called directly like a stored procedure.
- The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table.
- In contrast, a stored procedure must be called explicitly.
- Generally, triggers are of two types according to the SQL standard,
    - Row-level triggers
    - Statement-level triggers

Row-Level Trigger:

- It is activated for each row by a triggering statement such as insert, update, or delete.
- For example, if a table has inserted, updated, or deleted multiple rows, the row trigger is fired automatically for each row affected by the insert, update, or delete statement.

Statement-Level Trigger:

- It is a trigger, which is fired once for each event that occurs on a table regardless of how many rows are inserted, updated, or deleted.
- But MySQL doesn't support statement level trigger, it support only row level trigger.

18.2 WHY WE USE TRIGGERS

- Triggers help us to enforce business rules.
- Triggers help us to validate data even before they are inserted or updated.
- Triggers help us to keep a log of records like maintaining audit trails in tables.
- SQL triggers provide an alternative way to check the integrity of data.
- Triggers provide an alternative way to run the scheduled task.
- Triggers increases the performance of SQL queries because it does not need to compile each time the query is executed.

- Triggers reduce the client-side code that saves time and effort.
- Triggers help us to scale our application across different platforms.
- Triggers are easy to maintain.

18.3 Limitations of Triggers

- MySQL triggers do not allow to use of all validations; they only provide extended validations.
- For example, we can use the NOT NULL, UNIQUE, CHECK and FOREIGN KEY constraints for simple validations.
- Triggers are invoked and executed invisibly from the client application. Therefore, it isn't easy to troubleshoot what happens in the database layer.
- Triggers may increase the overhead of the database server.

18.4 Types of Events in Triggers

- We can define the maximum six types of actions or events in the form of triggers:
  - Before Insert
    - It is activated before the insertion of data into the table.
  - After Insert
    - It is activated after the insertion of data into the table.
  - Before Update
    - It is activated before the update of data in the table.
  - After Update
    - It is activated after the update of the data in the table.
  - Before Delete
    - It is activated before the data is removed from the table.
  - After Delete
    - It is activated after the deletion of data from the table.
- When we use a statement that does not use INSERT, UPDATE or DELETE query to change the data in a table, the triggers associated with the trigger will not be invoked.
- We have two type of key word which help use to manage old and new valued of table.
  - Old
    - It is temporary table created by system when called trigger.
    - It is available for update and delete event.
  - New
    - It is temporary table created by system when called trigger.
    - It is available for add and update event.

18.5 How To Create

Syntax:

```
Delimiter //
Create Trigger TriggerName
Before/After Insert/Update/Delete On TableName
For each row
Begin
    -- Sql statement
End //
Delimiter ;
```

Example:

```
Delimiter //
Create Trigger BeforeUpdateProduct
Before Update On Product
For each row
Begin
    Insert into UserLog (Activity, Description) values ("Update", Concat("User starting to
update ", + old.ProductName, " with " , new.ProductName));
End //
Delimiter ;

Delimiter //
Create Trigger AfterUpdateProduct
After Update On Product
For each row
Begin
    Insert into UserLog (Activity, Description) values ("Update", Concat(old.ProductName, "
updated successfully with ", new.ProductName ," by User"));
End //
Delimiter ;

-- Drop Trigger
Drop Trigger BeforeUpdateProduct;
Drop Trigger AfterUpdateProduct;

-- Show all triggers
SHOW TRIGGERS FROM triggerlearn;
```