MYSQL TUTORIAL NOTES

by: SIR MUHAMMAD FARHAN

Contact No: +92316-2859445

Email: mohammadfarhan44500@gmail.com

GitHub: https://github.com/muhammadfarhandevelper

SQL or MYSQL Tutorials

Data:

Data are individual facts, statistics, or items of information, often numeric.

Information:

Information is processed, organized and structured data.

Database:

A database is a collection of information that is organized so that it can be easily accessed, managed and updated.

OR

A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

Database Table:

Tables are database objects that contain all the data in a database. In tables, data is logically organized in a row-and-column format similar to a spreadsheet. Each row represents a unique record, and each column represents a field in the record.

ROW: A row represents a records.

Column: Column represents a field.

What is SQL?

SQL is a standard language for storing, manipulating and retrieving data in databases.

OR:

SQL language used in programming and designed for managing data held in a relational database management system

SQL stands for Structured Query Language.

SQL lets you access and manipulate databases.

SQL is an ANSI (American National Standards Institute) Standard

What can SQL DO?

- SQL can execute queries against a database.
- SQL can retrieve data from a database.
- SQL can insert records in a database.
- SQL can update records in a database.
- SQL can delete records in a database.
- SQL can create new table in a database.

Basically we can CRUD perform thought SQL (Structured Query Language).

C: (Create)

R: (Read)

U: (Update)

D: (Delete)

Data Management:

- Data management deals with managing large amount of information, which involves:
- the storage of information
- the provision of mechanisms for the manipulation of information
- providing safety of information stored under various circumstances

The two different approaches of managing data are as follows:

1. File-based systems
2. Database systems

File-Based Systems:

In a file-based systems data is stored in discrete files and a collection of such files is stored on a computer.

Rows in the table were called records and columns were called fields.

Disadvantages of File-Based Systems:

- Data redundancy and inconsistency
- Data isolation
- Concurrent access anomalies
- Security problems
- Integrity problems

Database Systems:

- Database Systems evolved in the late 1960s to address common issues in applications handling large volumes of data, which are also data intensive.
- At any point of time, data can be retrieved from the database, added, and searched based on some criteria in these databases.
- Databases are used to store data in an efficient and organized manner. A database allows quick and easy management of data.
- Data stored in this form is not permanent. Records in such manual files can only be maintained for a few months or few years

Advantages of Database Systems:

- The amount of redundancy in the stored data can be reduced
- No more inconsistencies in data
- The stored data can be shared
- Standards can be set and followed
- Data Integrity can be maintained. (no repetition and null values)
- Security of data can be implemented

What is Database Table?

- Database table is a tabular format and it is used to store the data of our application, software, organization etc.

- Table is made of rows and columns.
- A row represents a record.
- A column represents a field.

What is Entity?

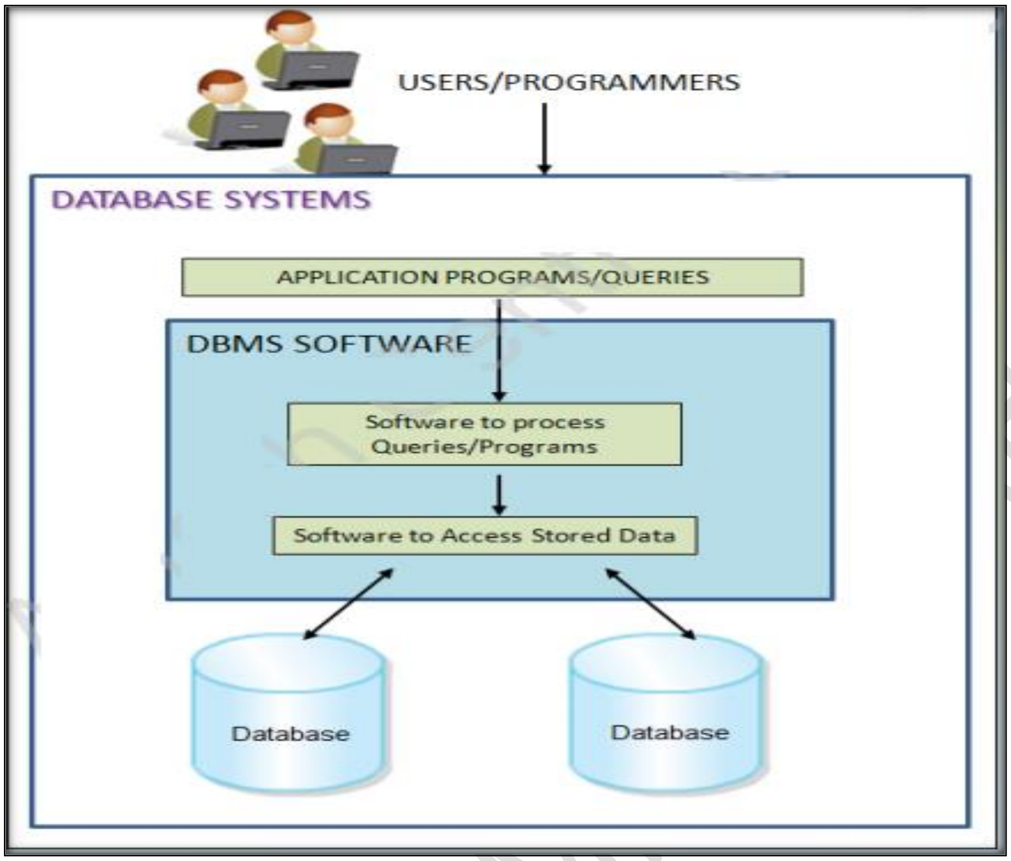
- An entity is something that exists as itself, as a subject or as an object.
- An entity is a person, place, thing, object, event, or even a concept, which can be distinctly identified.
- Each entity has certain characteristics known as attributes.
- For example, the student entity might include attributes like student number, name, and grade. Each attribute should be named appropriately.
- **For example:** the entities in a university are students, faculty members, and courses.
- A grouping of related entities becomes an entity set. Each entity set is given a name. The name of the entity set reflects the contents.

DATABASE MANAGEMENT SYSTEM (DBMS):

- A DBMS is a collection of related records and a set of programs that access and manipulate these records and enables the user to enter, store, and manage data.
- A database is a collection of interrelated data, and a DBMS is a set of programs used to add or modify this data.
- DBMS supports one of the four database models

Examples:

- Computerized library systems
- Flight reservation systems
- Automated teller machines ATM
- Computerized parts inventory systems


USERS/PROGRAMMERS
DATABASE SYSTEMS
APPLICATION PROGRAMS/QUERIES
DBMS SOFTWARE
Software to process Queries/Programs
Software to Access Stored Data
Database
Database

DATABASE MODELS:

- Databases can be differentiated based on functions and model of the data.
- The analysis and design of data models has been the basis of the evolution of databases.
- Each model has evolved from the previous one. The commonly used Database Models are as follows:
- A data model describes a container for storing data, and the process of storing and retrieving data from that container.

There are Four Database models:

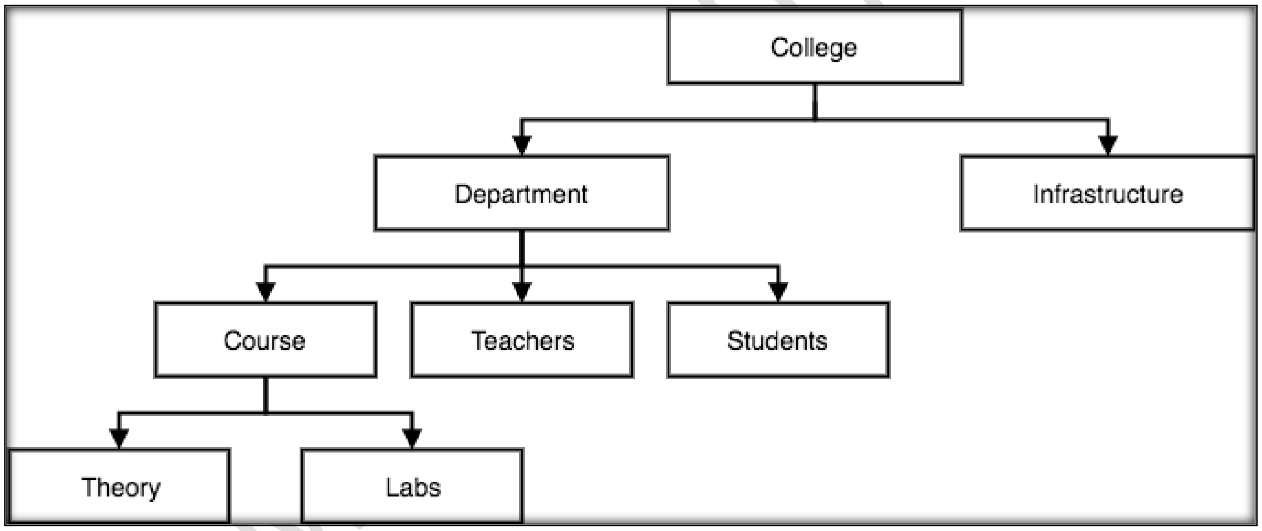
- Flat-file Data Model
- Hierarchical Data Model
- Network Data Model
- Relational Data Model

Flat-file Data Model:

- In this model, the database consists of only one table or file.
- This model is used for simple databases - for example, to store the roll numbers, names, subjects, and marks of a group of students.
- This model cannot handle very complex data. It can cause redundancy when data is repeated more than once.

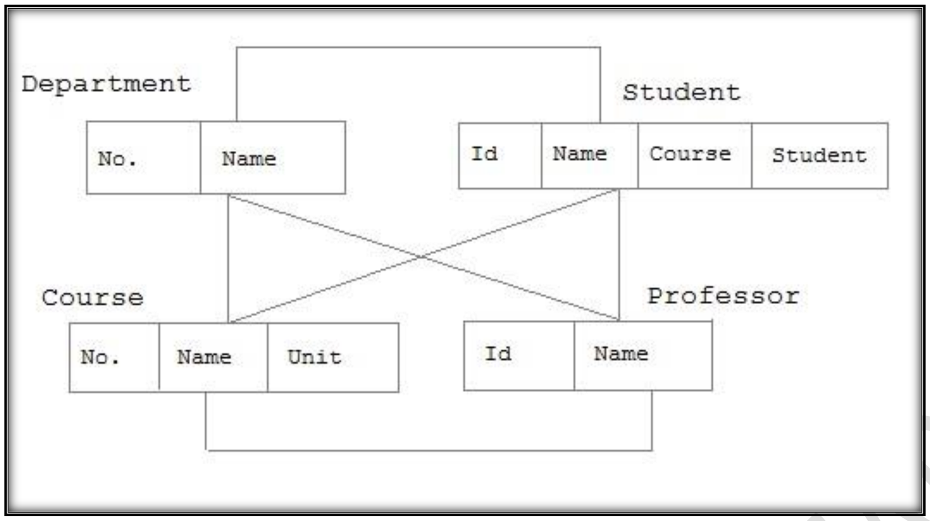
Hierarchical Data Model:

- In this model, different records are inter-related through hierarchical or tree-like structures.
- A parent record can have several children, but a child can have only one parent.
- To find data stored in this model, the user needs to know the structure of the tree.



NETWORK DATA MODEL:

- This model was developed to overcome the problems of hierarchical model.
- This model is similar to the Hierarchical Data Model. It is actually a subset of the network model.
- The set theory of the network model does not use a single-parent tree hierarchy. It allows a child to have more than one parent. Thus, the records are physically linked.



The Advantages of such a structure are specified as follows:

- Relationships are easier to implement in the network database model than in the hierarchical model.
- This model enforces database integrity.
- This model achieves sufficient data independence.

The Disadvantages are specified as follows:

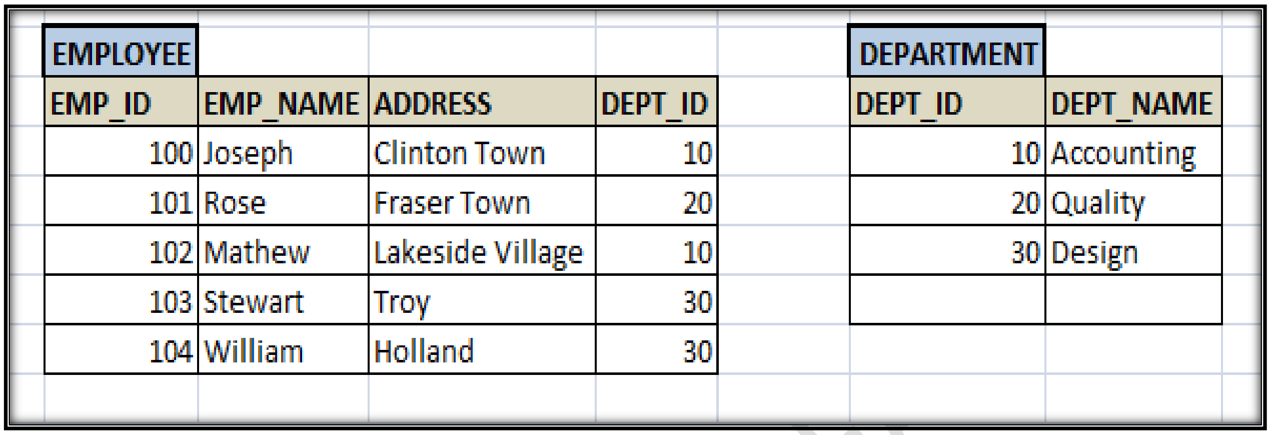
- The databases in this model are difficult to design.
- The programmer has to be familiar with the internal structures to access the database.

RELATIONAL DATA MODEL:

Edgar Frank Codd (19 August 1923 – 18 April 2003) was an English computer scientist who, while working for IBM, invented the relational model for database management in 1969.

- In this relational data model where all data is represented in terms of tuples (rows), grouped into relations (tables).
- The term **'Relation'** is derived from the set theory of mathematics. In the Relational Model, unlike the Hierarchical and Network models, there are no physical links.
- All data is maintained in the form of tables consisting of rows and columns. Data in two tables is related through common columns and not physical links.

- This led to the development of what came to be called the Relational Model database.
- Operators are provided for operating on rows in tables. This model represents the database as a collection of relations.



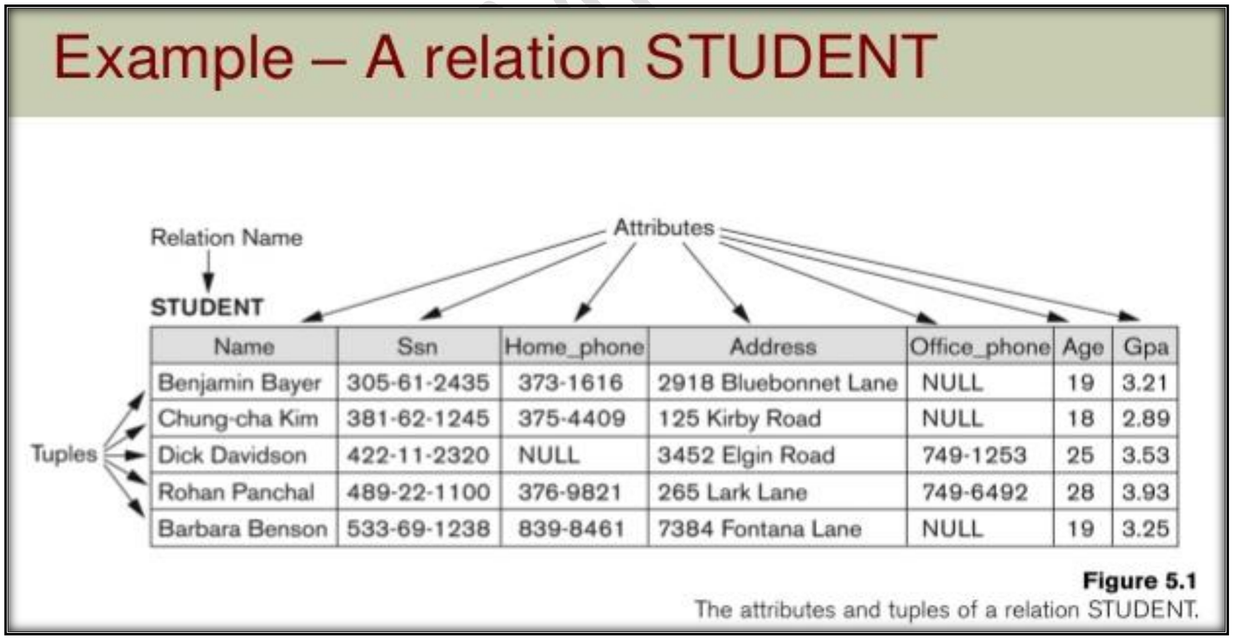
EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEPT_ID
100	Joseph	Clinton Town	10
101	Rose	Fraser Town	20
102	Mathew	Lakeside Village	10
103	Stewart	Troy	30
104	William	Holland	30

DEPARTMENT

DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

- A row is called a tuple or record.
- A column is called an attribute.
- The table is called a relation.
- Several attributes can belong to the same domain.



Example – A relation STUDENT

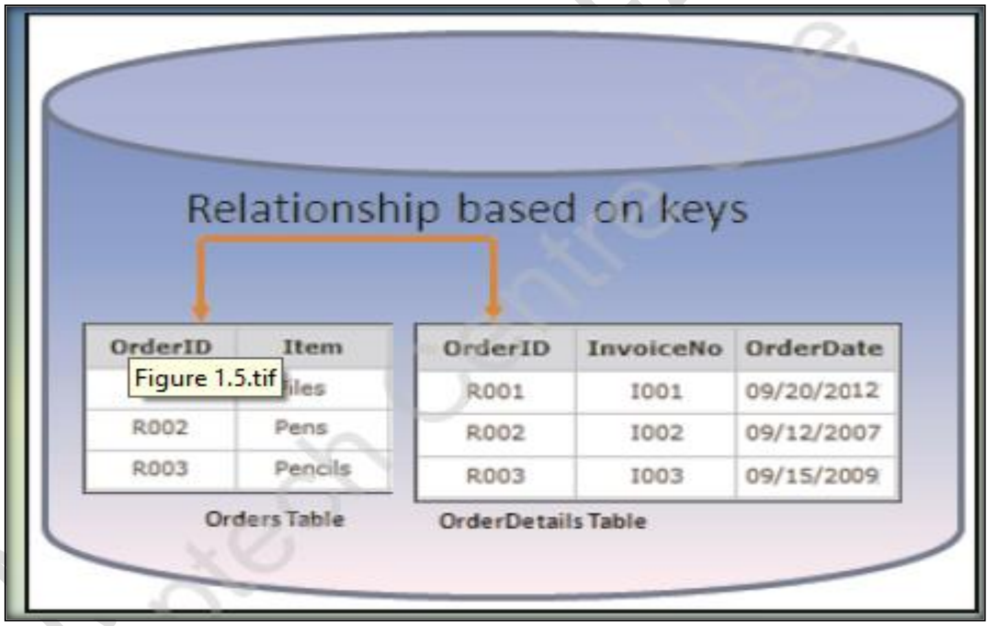
STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS):

- Relational Model is an attempt to simplify database structures.
- An RDBMS is a software program that helps to create, maintain, and manipulate a relational database.
- A relational database is a database divided into logical units called tables, where tables are related to one another within the database.
- LOGICAL AND PHYSICAL.
- Represents all data in the database as simple row-column tables of data values.
- Tables are related in a relational database, allowing adequate data to be retrieved in a single query (although the desired data may exist in more than one table). Example: SQL Server, MySQL, oracle is a RDBMS software.



OrderID	Item
[illegible]	iles
R002	Pens
R003	Pencils

OrderID	InvoiceNo	OrderDate
R001	I001	09/20/2012
R002	I002	09/12/2007
R003	I003	09/15/2009

SQL Statements:

For Integer or Numeric Data Types:

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types:

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Date and Time Data Types:

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999

smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

Character Strings Data Types:

Sr.No.	DATA TYPE & Description
1	**char** Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
2	**varchar** Maximum of 8,000 characters.(Variable-length non-Unicode data).
3	**varchar(max)** Maximum length of 2E + 31 characters, Variable-length non-Unicode data (SQL Server 2005 only).
4	**text** Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

The Basic and Running Data types:

1. int
2. big int
3. float
4. date
5. varchar
6. text

The SQL Create Database Statement:

The CREATE DATABASE statement is used to create a new SQL database.

Syntax:

CREATE DATABASE *databasename*;

The SQL Drop Database Statement:

The DROP DATABASE statement is used to drop an existing SQL database.

DROP DATABASE *databasename*;

The SQL Create Table Statement:

The CREATE TABLE statement is used to create a new table in a database.

Syntax:

```
create table table_name(
id int,
f_name varchar(20),
l_name varchar(20),
age int
)
```

The SQL Drop Table Statement:

Syntax:

DROP TABLE *table_name*;

The SQL Truncate Table Statement:

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax:

TRUNCATE TABLE *table_name*;

The SQL Insert Table Statement:

The INSERT INTO statement is used to insert new records in a table.

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

Syntax:

INSERT INTO *table_name* (*column1*, *column2*, *column3*,...)
VALUES (*value1*, *value2*, *value3*, ...);

Example:

INSERT INTO Customers(CustomerName, ContactName, Address, City, PostalCode, Country)VALUES ('Cardinal', 'TomB.Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

2.Not Specify the column names:

Syntax:

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

Example:

INSERT INTO usertbl VALUES ('[value-1]','[value-2]','[value-3]')

The SQL Select Table Statement:

The SELECT statement is used to select data from a database.

Syntax:

SELECT * FROM *table_name*;

* means ALL.

Or

Specify some column

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

The SQL Select Distinct Statement:

The SELECT DISTINCT statement is used to return only distinct (different) values.

Syntax:

SELECT DISTINCT *column1*, *column2, ...*
FROM *table_name*;

Example:

SELECT DISTINCT * FROM usertbl

The SQL Where Clause:

The WHERE clause is used to **filter records.**

Syntax:

SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition*;

Example:

SELECT * FROM Customers
WHERE Country='Mexico'

OR

WHERE CustomerID=1;

Operators in The WHERE Clause

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Example of Between:

```
SELECT * FROM Products
WHERE Price BETWEEN 50 AND 60;
```

Example of Like:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length

WHERE ContactName LIKE 'u%a'	Finds any values that start with "u" and ends with "a"

Example of IN:

SELECT * FROM Customers

WHERE City IN ('Paris','London');

The SQL AND OR and NOT Operators:

AND EXAMPLE:

SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';

OR EXAMPLE:

SELECT * FROM Customers
WHERE City='Berlin' OR City='München';

NOT EXAMPLE:

SELECT * FROM Customers
WHERE NOT Country='Germany';

Combining AND, OR and NOT

Example:

SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');

Another Example:

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

The SQL Order by Keyword:

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

Example (Ascending):

```
SELECT * FROM Customers
ORDER BY Country;
```

Example (Descending):

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

Another Example:

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

SQL Update Statement:

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

Syntax:

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

Example:

```
UPDATE Customers
SET ContactName = 'Arif Alvi', City= 'Frankfurt'
WHERE CustomerID = 1;
```

Here where clause is mandatory if you will not write where condition so all data will have updated.

SQL DELETE STATEMENT

The DELETE statement is used to delete existing records in a table.

Syntax:

```
DELETE FROM table_name WHERE condition;
```

Here where clause is mandatory if you will not write where condition so all data will have deleted.

Example:

```
DELETE FROM Customers WHERE CustomerName='ali';
```

Aggregate Functions:

SQL MIN(), MAX(), COUNT , AVG,SUM FUNCTIONS

Min()

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

Example:

```
Select min(price) from product;
```

Max()

```
SELECT MAX(Price) FROM product;
```

COUNT()

```
SELECT COUNT(P_ID) FROM products;
```

AVG()

```
SELECT AVG(Price) FROM Products;
```

SUM()

SELECT SUM(price) FROM products;

SQL LIMIT Clause:

Syntax:

SELECT * FROM Customers LIMIT 3;

Here 3 denoted three first rows.

And other syntax with two parameter

Syntax:

SELECT * FROM Customers LIMIT 2,6;

Here 2 denoted where you start. It is also called offset.
Here 6 is denoted my limit how many rows you want

CONSTRAINTS:

Constraints are the rules enforced on the data columns of a 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.

SQL NOT NULL CONTSTRAINTS:

Ensures that a column cannot have NULL value.

Example:

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

SQL Default CONTSTRAINTS:

Provides a default value for a column when none is specified.

Example:

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   SALARY   DEFAULT 5000,
);
```

SQL UNIQUE CONTSTRAINTS:

Ensures that all values in a column are different.

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

SQL CHECK CONTSTRAINTS:

The CHECK constraint ensures that all the values in a column satisfies certain conditions.

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL CHECK (AGE >= 18),
   ADDRESS  CHAR (25)
);
```

SQL PRIMARY KEY CONTSTRAINTS:

A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.

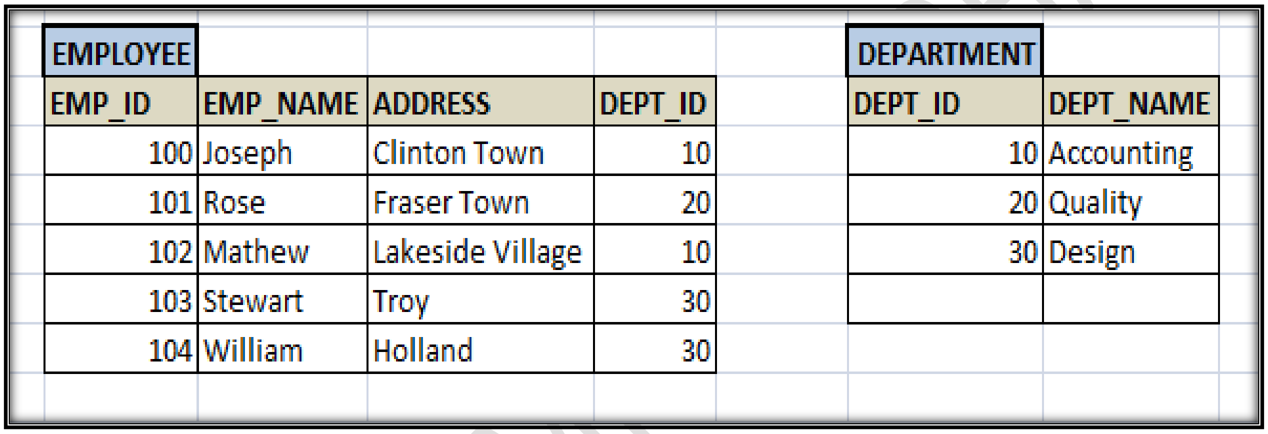
```
CREATE TABLE CUSTOMERS(
   ID   INT  primary key,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25)
);
```

SQL FOREIGN KEY CONTSTRAINTS:

Uniquely identifies a row / record in another table.

Primary and Foreign key use with each other for create a relation in a tables.

RELATION:



EMPLOYEE			
EMP_ID	EMP_NAME	ADDRESS	DEPT_ID
100	Joseph	Clinton Town	10
101	Rose	Fraser Town	20
102	Mathew	Lakeside Village	10
103	Stewart	Troy	30
104	William	Holland	30

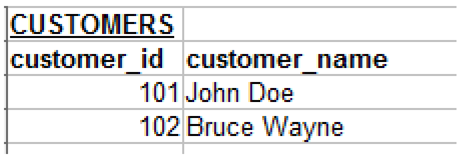
DEPARTMENT	
DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

Syntax:

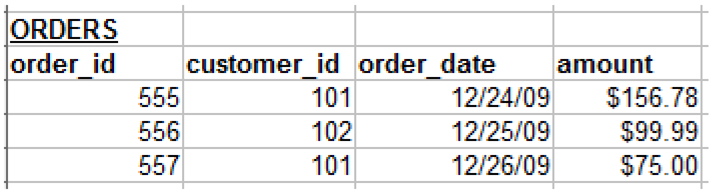
```
CREATE TABLE CUSTOMERS(
  C_ID   INT  primary key auto_increment,
   C_NAME VARCHAR (20)     NOT NULL,
   C_AGE  INT              NOT NULL,
   C_ADDRESS  CHAR (25)
);
```

```
CREATE TABLE Orders(
   O_ID   INT  primary key,
   O_NAME VARCHAR (20)     NOT NULL,
  C_ID int, foreign key (C_ID) references CUSTOMERS(C_ID)
);
```

Examples:

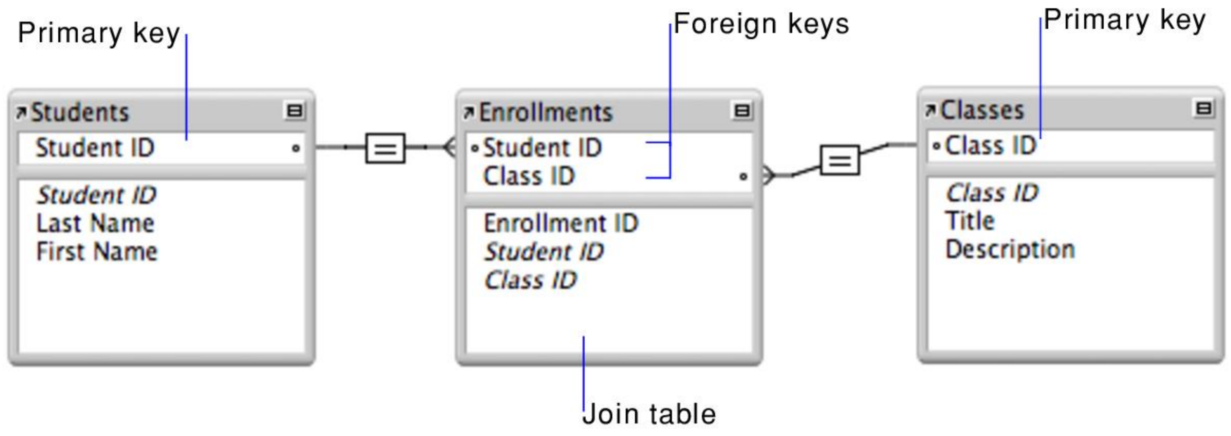


CUSTOMERS	
customer_id	customer_name
101	John Doe
102	Bruce Wayne



ORDERS			
order_id	customer_id	order_date	amount
555	101	12/24/09	$156.78
556	102	12/25/09	$99.99
557	101	12/26/09	$75.00

Another Example:



Alias for Columns:

The following SQL statement creates two aliases, one for the CustomerID column and one for the CustomerName column:

Example:

SELECT CustomerID AS ID,CustomerName AS Customer
FROM Customers;

Alias for Table:

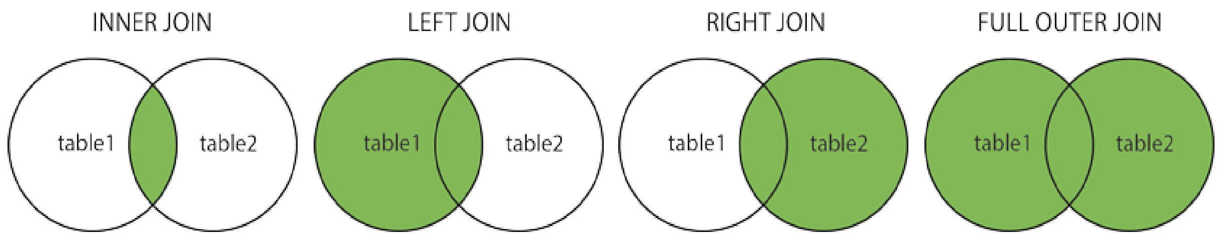
SELECT *column_name(s)*
FROM *table_name* AS *alias_name;*

JOINS:

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

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



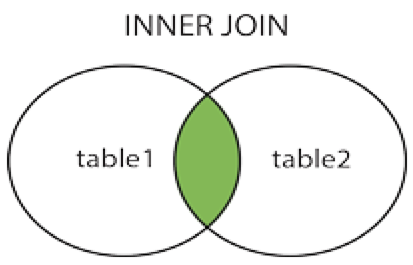
INNER JOINS:

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

Example:

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

Foreign Key **Primary key**



General Example:

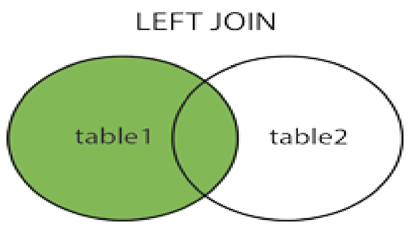
select * from employee as em INNER JOIN department as dp on em.dept_id = dp.dept_id

Join Three Tables:

SELECT Orders.OrderID,Customers.CustomerName,Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

LEFT JOINS:

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).



Syntax:

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

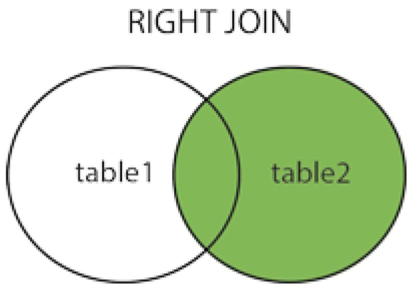
Example:

SELECT Customers.CustomerName,Orders.OrderID
FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;

Note: The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

RIGHT JOINS:

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).



Syntax:

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

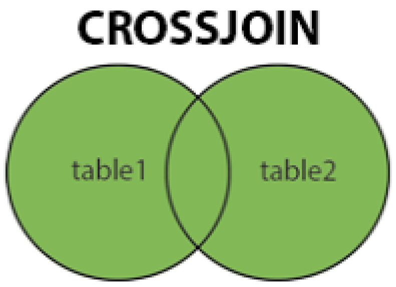
Example:

SELECT Orders.OrderID,Employees.LastName,Employees.FirstName
FROM Orders RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID ORDER BY Orders.OrderID;

Note: The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

CROSS JOINS:

The CROSS JOIN keyword returns all records from both tables (table1 and table2).



Syntax:

SELECT *column_name(s)*FROM *table1*
CROSS JOIN *table2*;

Example:

SELECT Customers.CustomerName,Orders.OrderID FROM Customers
CROSS JOIN Orders;

Note:

If you add a WHERE clause (if table1 and table2 has a relationship), the CROSS JOIN will produce the same result as the INNER JOIN clause:

Another Example:

SELECT Customers.CustomerName,Orders.OrderID
FROM CustomersCROSS JOIN Orders
WHERE Customers.CustomerID=Orders.CustomerID;

Group by Statement:

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

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

Syntax:

SELECT *column_name(s)*
FROM *table_name*
GROUP BY *column_name(s)*

Example:

SELECT COUNT(CustomerID), Country FROM Customers
GROUP BY Country;

MYSQL Having Clause:

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

Syntax:

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

Example:

SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country
HAVING COUNT(CustomerID) > 5;

UNION JOINS:

The UNION operator is used to combine the result-set of two or
more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

Syntax:

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

Example:

SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;

UNION ALL Syntax:

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

Example:

SELECT City FROM Customers UNION ALL SELECT City FROM Suppliers
ORDER BY City;

SQL UNION WITH WHERE:

Example:

SELECT City, Country FROM Customers WHERE Country='Germany' UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;

Sub Query or Nested Query?

A MySQL subquery is a query nested within another query such
as SELECT, INSERT, UPDATE or DELETE. Also, a subquery can be nested within another subquery.

A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query.

Example:

select * from employee where c_id = (select c_id from city where name = 'karachi');

Exist and Not Exist?

select * FROM employee where EXISTS (SELECT c_id FROM city where name = 'hyd');

Not Exist:

select * FROM employee where not EXISTS (SELECT c_id FROM city where name = 'hyd');

MY SQL Alter Statement:

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

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

Alter Table Add Column:

Syntax:

ALTER TABLE *table_name*
ADD *column_name datatype*;

Example:

Alter table employee add email varchar(20) not null

Alter Table add column with Order:

Alter table employee add email varchar(20) after name;

Alter Table change Data type of column:

Alter table employee modify email int(10);

Alter Table Drop Column:

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

Example:

ALTER TABLE employee
DROP COLUMN Email;

Alter Table Rename Column Name:

ALTER TABLE *table_name*
CHANGE email *email _address varchar(20)*;

Alter Table Modify Column constraint:

Syntax:

ALTER TABLE *table_name*
ADD UNIQUE(email) ;

Alter Table Add Constraint in Existing Column:

ALTER TABLE employee ADD FOREIGN KEY (c_id) REFERENCES city(c_id);

Alter Table Change Table Name:

ALTER TABLE employee rename employee_tbl;

IF Clause:

The IF() function returns a value if a condition is TRUE, or another value if a condition is FALSE.

Syntax:

IF(*condition*, *value_if_true*, *value_if_false*)

Example:

SELECT OrderID, Quantity, IF(Quantity>10, "MORE", "LESS")
FROM OrderDetails;

Another Example:

SELECT P_name , price , if(price>2000,"High Rate Product","Low Rate Product") as Stage from product

Case Clause:

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

Example:

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

Another Example:

```
SELECT P_id, price,

CASE

    WHEN price > 4000 THEN 'The quantity is greater than All'

    WHEN price = 2000 THEN 'The quantity is 2000'

    ELSE 'The quantity is under poor ( only for gareeb log)'

END AS Price_status FROM product;
```

Another Example:

```
SELECT id,stname,percentage,

CASE

WHEN (percentage>=90 and percentage<=100) THEN "A+1"

WHEN (percentage>=80 and percentage<90) THEN "A1"

WHEN (percentage>=70 and percentage<80) THEN "A"

ELSE "FAIL"

END as Grade
```

FROM studentresult

MYSQL Numeric Functions:

Function	Description
ABS	Returns the absolute value of a number
CEIL	Returns the smallest integer value that is >= to a number
DIV	Used for integer division
FLOOR	Returns the largest integer value that is <= to a number
MOD	Returns the remainder of a number divided by another number
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of another number
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places
SQRT	Returns the square root of a number

MySQL String Functions:

Function	Description
CHAR_LENGTH	Returns the length of a string (in characters)
CHARACTER_LENGTH	Returns the length of a string (in characters)
LENGTH	Returns the length of a string (in bytes)
CONCAT	Adds two or more expressions together
CONCAT_WS	Adds two or more expressions together with a separator
FORMAT	Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places. Ex: format(45000350.45400)
STRCMP	Compares two strings
LTRIM	Removes leading spaces from a string
RTRIM	Removes trailing spaces from a string
TRIM	Removes leading and trailing spaces from a string
LOWER	Converts a string to lower-case
LCASE	Converts a string to lower-case
UCASE	Converts a string to upper-case

UPPER	Converts a string to upper-case

DATE & TIME FUNCTION:

Function	Description
ADDDATE	Adds a time/date interval to a date and then returns the date EX: ADDDATE(“2012-03-23”,Interval 3 Month)
ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
SYSDATE	Returns the current date and time
NOW	Returns the current date and time
CURRENT_TIME	Returns the current time
CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time

DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
SUB**DATE**	Subtracts a time/date interval from a date and then returns the date
DAY	Returns the day of the month for a given date
HOUR	Returns the hour part for a given date
LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MINUTE	Returns the minute part of a time/datetime
MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
QUARTER	Returns the quarter of the year for a given date value
SECOND	Returns the seconds part of a time/datetime
TIMEDIFF	Returns the difference between two time/datetime expressions
TIMESTAMP	Returns a datetime value based on a date or datetime value
YEAR	Returns the year part for a given date

User Defined Function:

User Defined Function is the code that extends the functionality of MySQL server by adding external code can work same as inbuilt functions like concat(), length() in MySQL. User-defined functions are compiled as object files which can be added with statement CREATE FUNCTION and can be removed from the server with statement DROP FUNCTION dynamically.

User Defined functions are useful when you want to extend the functionalities of your MySQL server.

1. User-defined functions take zero or more input parameters, and return a single value such as a string, integer, or real values.
2. You can define simple function that operate on a single row at a time or an aggregate functions that operate on groups of rows.
3. You can indicate that a function returns NULL or that an error occurred.

User defined function syntax is very similar to stored procedures in MySQL. Here I have created simple user-defined functions which are to calculate available credits in the user account.

Example (Non Parameterized Function):

```
DELIMITER //
CREATE FUNCTION showdata()
RETURNS varchar(20)
BEGIN

        RETURN 'WELCOME TO OUR WORLD';

END //
DELIMITER;
```

Another Example (Parameterized Function):

```
DELIMITER //
CREATE FUNCTION showName(fname varchar(20) , lname varchar(20))
RETURNS varchar(40)
BEGIN
        RETURN concat(fname, ' ' ,lname);
```

```
END //
DELIMITER;
```

Real Time Example (Calculate Age):

```
DELIMITER //
CREATE FUNCTION CalculateAge(dob date)
RETURNS int
BEGIN
        Declare today_date date;
        Select current_date() into today_date;
        RETURN year(today_date) – year(dob);

END //
DELIMITER ;
```

Introduction to MySQL Queries:

There are many kinds of SQL commands which can be categorized into the following:

- DDL (Data definition language)
- DML (Data manipulation language)
- DQL (Data query language)
- DCL (Data control language)
- TCL (Transaction control language)

DDL (Data Definition Language):

When we perform any changes with the physical structure of the table in the database, then we need DDL commands. CREATE, ALTER, RENAME, DROP, TRUNCATE, etc commands come into this category. Those commands can't be rolled back.

DML (Data Manipulation Language):

As we can see the name Data Manipulation language, so once the tables/database are created, to manipulate something inside that stuff we require DML commands. Merits of using these commands are if incase any wrong changes happened, we can roll back/undo it.

Example:

INSERT, UPDATE, DELETE etc.

DCL (Data Control Language):

It grants or revokes access of users to the database.

Example:

GRANT CREATE table to user;

```
REVOKE CREATE table from user;
```

TCL (Transaction Control Language):

This manages the issues related to the transaction in any database. This is used to rollback or commit in the database.

Example:

```
ROLLBACK;

COMMIT;
```

DQL (Data Query Language):

Data query language consists of only **SELECT** command by which we can retrieve and fetch data on the basis of some conditions provided. Many clauses of SQL are used with this command for retrieval of filtered data.

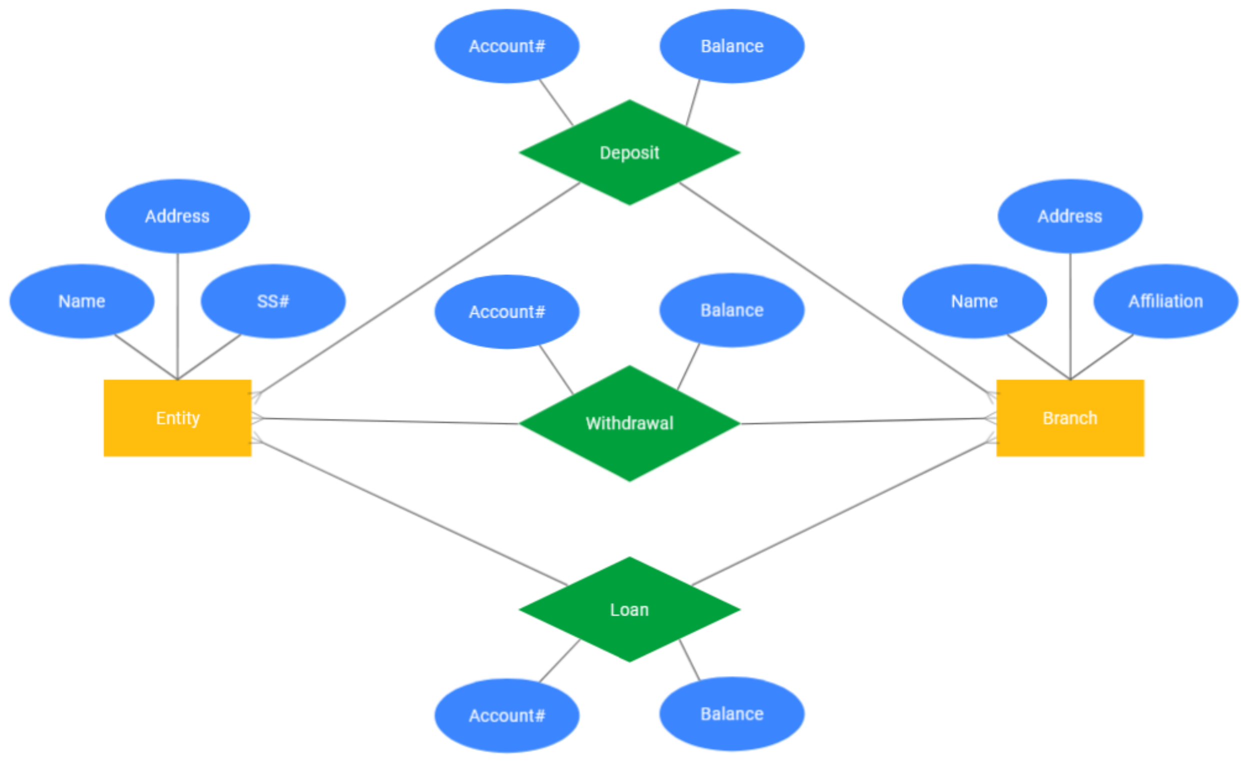
Conclusion:

These commands and clauses we have discussed above are very useful in real-time scenarios as it provides the basic concepts of how to use SQL queries to fetch and manipulate data in the database. Apart from this, while using advance and analytical queries like window function etc, these clauses are very important.

ER (Entity Relationship) Diagram:

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.



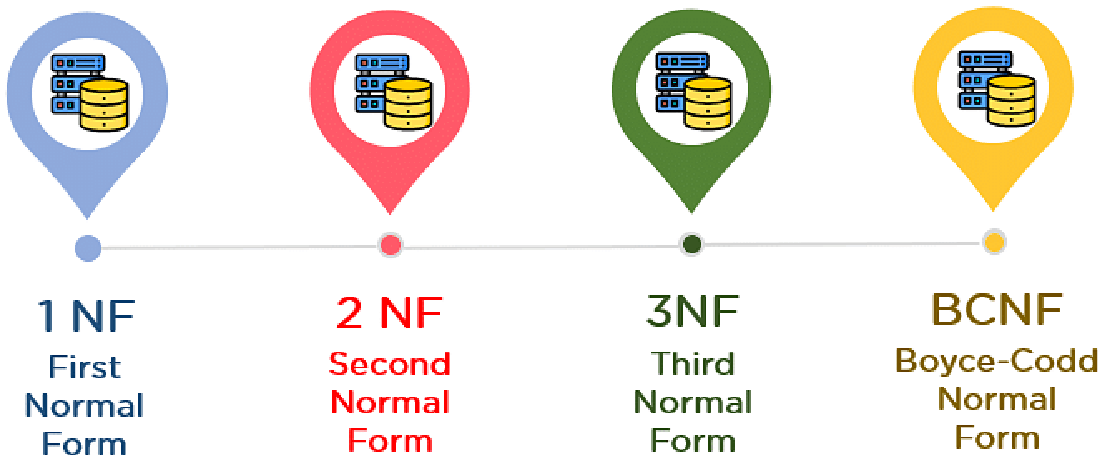
There are three main types of relationships in a database expressed using cardinality notation in an ER diagram.

- one-to-one
- one-to-many
- many-to-many

What is Normalization?

Normalization is the process to eliminate data redundancy and enhance data integrity in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.

Normalization organizes the columns and tables of a database to ensure that database integrity constraints properly execute their dependencies. It is a systematic technique of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update, and Deletion anomalies.



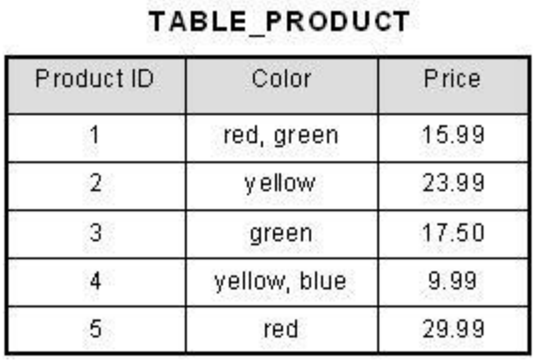
1st Normal Form (1NF):

- A table is referred to as being in its First Normal Form if atomicity of the table is 1.

- Here, atomicity states that a single cell cannot hold multiple values. It must hold only a single-valued attribute.
- The First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

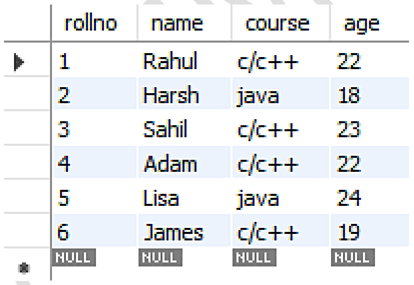
Now you will understand the First Normal Form with the help of an example.

Below is a students' record table that has information about student roll number, student name, student course, and age of the student.



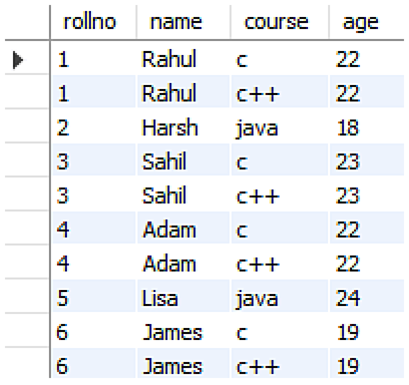
TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99



rollno	name	course	age
1	Rahul	c/c++	22
2	Harsh	java	18
3	Sahil	c/c++	23
4	Adam	c/c++	22
5	Lisa	java	24
6	James	c/c++	19
NULL	NULL	NULL	NULL

In the studentsrecord table, you can see that the course column has two values. Thus it does not follow the First Normal Form. Now, if you use the First Normal Form to the above table, you get the below table as a result.



rollno	name	course	age
1	Rahul	c	22
1	Rahul	c++	22
2	Harsh	java	18
3	Sahil	c	23
3	Sahil	c++	23
4	Adam	c	22
4	Adam	c++	22
5	Lisa	java	24
6	James	c	19
6	James	c++	19

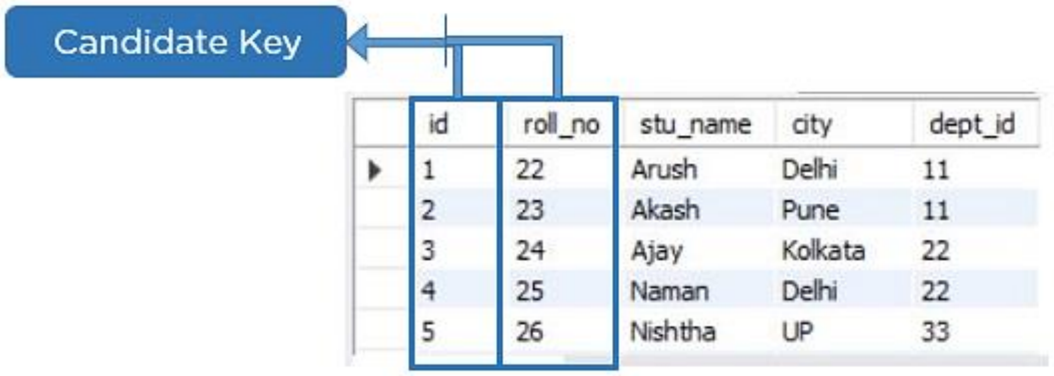
By applying the First Normal Form, you achieve atomicity, and also every column has unique values.

Before proceeding with the Second Normal Form, get familiar with Candidate Key and Super Key.

Candidate Key

A candidate key is a set of one or more columns that can identify a record uniquely in a table, and YOU can use each candidate key as a Primary Key.

Now, let’s use an example to understand this better.

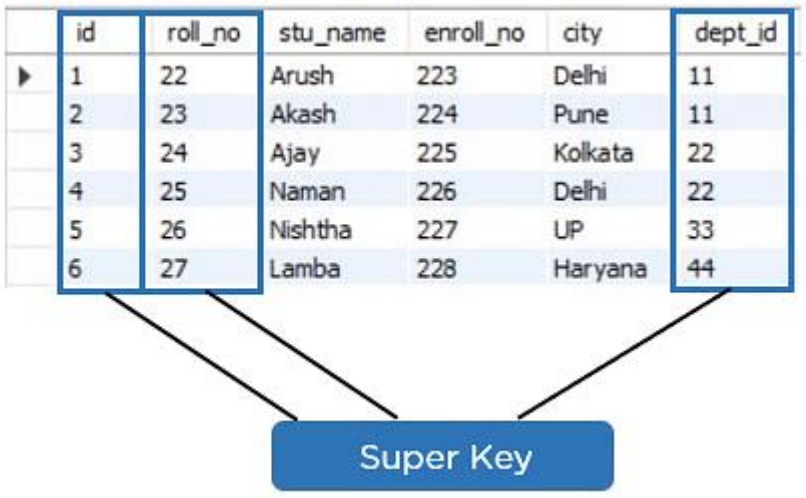

Candidate Key

id	roll_no	stu_name	city	dept_id
1	22	Arush	Delhi	11
2	23	Akash	Pune	11
3	24	Ajay	Kolkata	22
4	25	Naman	Delhi	22
5	26	Nishtha	UP	33

Super Key:

Super key is a set of over one key that can identify a record uniquely in a table, and the Primary Key is a subset of Super Key.

Let’s understand this with the help of an example.



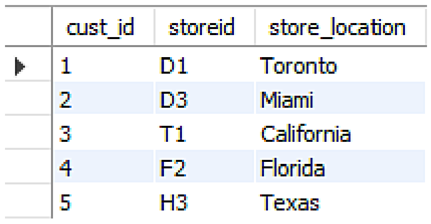
id	roll_no	stu_name	enroll_no	city	dept_id
1	22	Arush	223	Delhi	11
2	23	Akash	224	Pune	11
3	24	Ajay	225	Kolkata	22
4	25	Naman	226	Delhi	22
5	26	Nishtha	227	UP	33
6	27	Lamba	228	Haryana	44

Second Normal Form (2NF):

The first condition for the table to be in Second Normal Form is that the table has to be in First Normal Form. The table should not possess partial dependency. The partial dependency here means the proper subset of the candidate key should give a non-prime attribute.

Now understand the Second Normal Form with the help of an example.

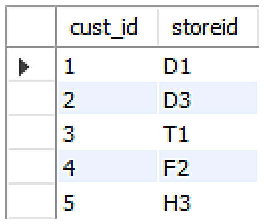
Consider the table Location:



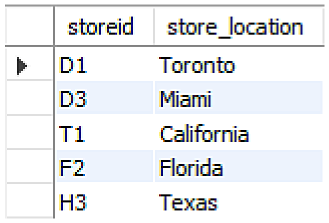
cust_id	storeid	store_location
1	D1	Toronto
2	D3	Miami
3	T1	California
4	F2	Florida
5	H3	Texas

The Location table possesses a composite primary key cust_id, storeid. The non-key attribute is store_location. In this case, store_location only depends on storeid, which is a part of the primary key. Hence, this table does not fulfill the second normal form.

To bring the table to Second Normal Form, you need to split the table into two parts. This will give you the below tables:



cust_id	storeid
1	D1
2	D3
3	T1
4	F2
5	H3



storeid	store_location
D1	Toronto
D3	Miami
T1	California
F2	Florida
H3	Texas

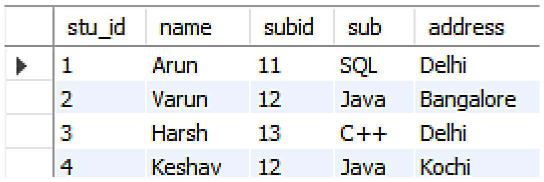
As you have removed the partial functional dependency from the location table, the column store_location entirely depends on the primary key of that table, storeid.

Now that you understood the 1st and 2nd Normal forms, you will look at the next part of this Normalization in SQL tutorial.

Third Normal Form (3NF):

- The first condition for the table to be in Third Normal Form is that the table should be in the Second Normal Form.
- The second condition is that there should be no transitive dependency for non-prime attributes, which indicates that non-prime attributes (which are not a part of the candidate key) should not depend on other non-prime attributes in a table. Therefore, a transitive dependency is a functional dependency in which A → C (A determines C) indirectly, because of A → B and B → C (where it is not the case that B → A).
- The third Normal Form ensures the reduction of data duplication. It is also used to achieve data integrity.

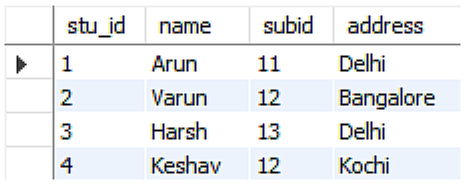
Below is a student table that has student id, student name, subject id, subject name, and address of the student as its columns.



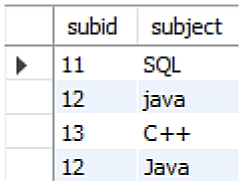
stu_id	name	subid	sub	address
1	Arun	11	SQL	Delhi
2	Varun	12	Java	Bangalore
3	Harsh	13	C++	Delhi
4	Keshav	12	Java	Kochi

In the above student table, stu_id determines subid, and subid determines sub. Therefore, stu_id determines sub via subid. This implies that the table possesses a transitive functional dependency, and it does not fulfill the third normal form criteria.

Now to change the table to the third normal form, you need to divide the table as shown below:



stu_id	name	subid	address
1	Arun	11	Delhi
2	Varun	12	Bangalore
3	Harsh	13	Delhi
4	Keshav	12	Kochi



subid	subject
11	SQL
12	java
13	C++
12	Java

As you can see in both the tables, all the non-key attributes are now fully functional, dependent only on the primary key. In the first table, columns name, subid, and addresses only depend on stu_id. In the second table, the sub only depends on subid.

Boyce CoddNormal Form (BCNF):

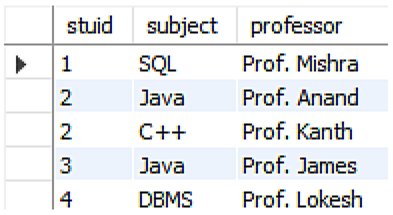
Boyce Codd Normal Form is also known as 3.5 NF. It is the superior version of 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to tackle certain types of anomalies which were not resolved with 3NF.

The first condition for the table to be in Boyce Codd Normal Form is that the table should be in the third normal form. Secondly, every Right-Hand Side (RHS) attribute of the functional dependencies should depend on the super key of that particular table.

For Example:

You have a functional dependency X → Y. In the particular functional dependency, X has to be the part of the super key of the provided table.

Consider the below subject table:



stuid	subject	professor
1	SQL	Prof. Mishra
2	Java	Prof. Anand
2	C++	Prof. Kanth
3	Java	Prof. James
4	DBMS	Prof. Lokesh

The subject table follows these conditions:

- Each student can enroll in multiple subjects.
- Multiple professors can teach a particular subject.
- For each subject, it assigns a professor to the student.

In the above table, student_id and subject together form the primary key because using student_id and subject; you can determine all the table columns.

Another important point to be noted here is that one professor teaches only one subject, but one subject may have two professors.

Which exhibit there is a dependency between subject and professor, i.e. subject depends on the professor's name.

to

The table is in 1st Normal form as all the column names are unique, all values are atomic, and all the values stored in a particular column are of the same domain.

The table also satisfies the 2nd Normal Form, as there is no Partial Dependency.

And, there is no Transitive Dependency; hence, the table also satisfies the 3rd Normal Form.

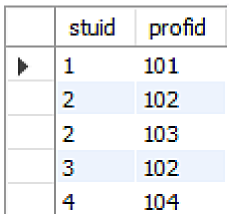
This table follows all the Normal forms except the Boyce Codd Normal Form.

As you can see stuid, and subject forms the primary key, which means the subject attribute is a prime attribute.

However, there exists yet another dependency - professor → subject.

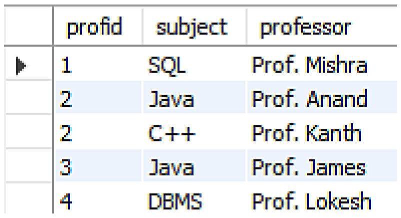
BCNF does not follow in the table as a subject is a prime attribute, the professor is a non-prime attribute.

To transform the table into the BCNF, you will divide the table into two parts. One table will hold stuid which already exists and the second table will hold a newly created column profid.



stuid	profid
1	101
2	102
2	103
3	102
4	104

And in the second table will have the columns profid, subject, and professor, which satisfies the BCNF.



profid	subject	professor
1	SQL	Prof. Mishra
2	Java	Prof. Anand
2	C++	Prof. Kanth
3	Java	Prof. James
4	DBMS	Prof. Lokesh

With this, you have reached the conclusion of the 'Normalization in SQL' tutorial.

Conclusion:

In this tutorial, you have seen Normalization in SQL and understood the different Normal forms of Normalization. Now, you can organize the data in the database and remove the data redundancy and promote data integrity. This tutorial also helps beginners for their interview processes to understand the concept of Normalization in SQL.

MYSQL CREATE VIEW STATEMENT:

In SQL, a 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.

A view is created with the CREATE VIEW statement.

Syntax:

CREATE VIEW *view_name* AS
SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition*;

MYSQL DROP VIEW STATEMENT:

Syntax:

DROP VIEW *view_name*;

MYSQL ALTER VIEW STATEMENT:

```
ALTER VIEW customerBrazil AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

MYSQL CREATE INDEX STATEMENT:

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes; they are just used to speed up searches/queries.

Create Index Syntax:

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

Example:

CREATE INDEX idx_lastname
ON Persons (LastName);

Create Unique Index Syntax:

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

Drop Index Statement:

DROP INDEX *index_name* ON tbl_name;

Extra SQL Queries:

Explain select query.

show index from table_name

describe (show the structured table)

MYSQL TRIGGERS:

A trigger in MySQL is a set of SQL statements stored in the database. **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 and statement-level triggers.

Row-Level Trigger: It is a trigger, which 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
- 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.

NOTE: We should know that MySQL doesn't support statement-level triggers. It provides supports for row-level triggers only.

TYPES OF TRIGGERS IN MYSQL:

- Before Insert – activated before data is inserted into the table
- After Insert -- activated after data is inserted into the table
- Before Update -- activated before data in the table is updated
- After Update -- activated after data in the table is updated
- Before Delete -- activated before data is removed from the table
- After Delete -- activated after data is removed from the table

Syntax:

CREATE TRIGGER Trigger_Name trigger_time trigger_event

ON [Table_Name]

FOR EACH ROW

Begin

…………..

END

Example:

```
Delimiter //
CREATE TRIGGER Trigger_Name AFTER Insert
ON student
FOR EACH ROW
Begin
Insert into st_audit values(null,concat('A row is inserted in student table at' , now()));
End //
Delimiter ;
```

NEW and OLD Keywords in MySQL:

- Mysql Triggers provide us 2 magical or virtual tables called NEW and OLD.
- When we inert a row in a table then that row is also inserted in NEW table.
- When we delete a row from a table then that row is also inserted in OLD table.

Auto Generator Id with Constant Character:

```
DELIMITER //
CREATE TRIGGER idgenerator BEFORE INSERT
on product FOR EACH ROW

BEGIN
     INSERT INTO pseqid VALUES(null);
   SET NEW.id = concat('PD-',LAST_INSERT_ID());
END //

DELIMITER ;
```

Transaction with ACID in MySQL:

A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit. In other words, a transaction will never be complete unless each individual operation within the group is successful. If any operation within the transaction fails, the entire transaction will fail.

Example:

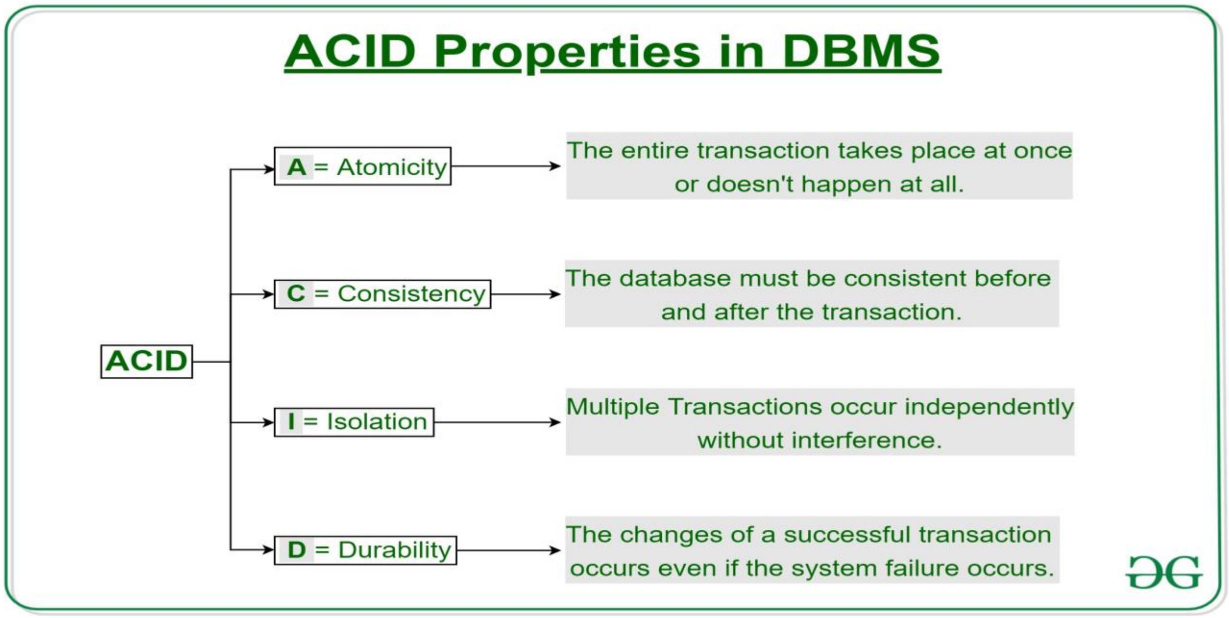
```
START TRANSACTION;
SELECT * FROM newstudent;
DELETE FROM newstudent;
SELECT  * FROM newstudent;
ROLLBACK;
SELECT * FROM newstudent;
```

Properties of Transaction in MySQL:

Transactions have the following four standard properties, usually referred to by the acronym **ACID** −

- **Atomicity** − This ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure and previous operations are rolled back to their former state.
- **Consistency** − This ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation** − This enables transactions to operate independently on and transparent to each other.
- **Durability** − This ensures that the result or effect of a committed transaction persists in case of a system failure.

In MySQL, the transactions begin with the statement **BEGIN WORK** and end with either a **COMMIT** or a **ROLLBACK** statement. The SQL commands between the beginning and ending statements form the bulk of the transaction.


ACID Properties in DBMS
ACID
A = Atomicity
The entire transaction takes place at once or doesn't happen at all.
C = Consistency
The database must be consistent before and after the transaction.
I = Isolation
Multiple Transactions occur independently without interference.
D = Durability
The changes of a successful transaction occurs even if the system failure occurs.