



SQL TUTORIAL NOTES

by: **SIR MUHAMMAD FARHAN**

Contact No: +92316-2859445

Email: mohammadfarhan44500@gmail.com

GitHub: <https://github.com/muhammadfarhandeveloper>

SQL 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 through 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?

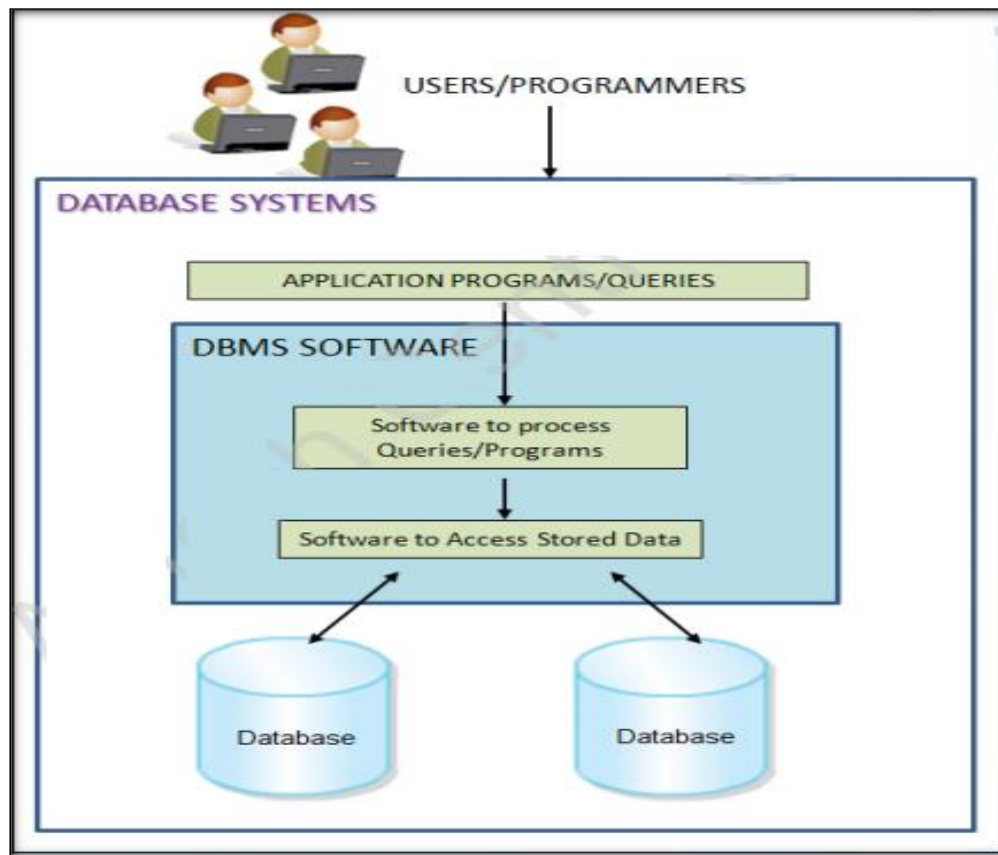
- An entity is something that exists as itself, as a subject or as an object.
- An entity is a person, place, thing, object, event, or even a concept, which can be distinctly identified.
- Each entity has certain characteristics known as attributes.
- For example, the student entity might include attributes 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



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:

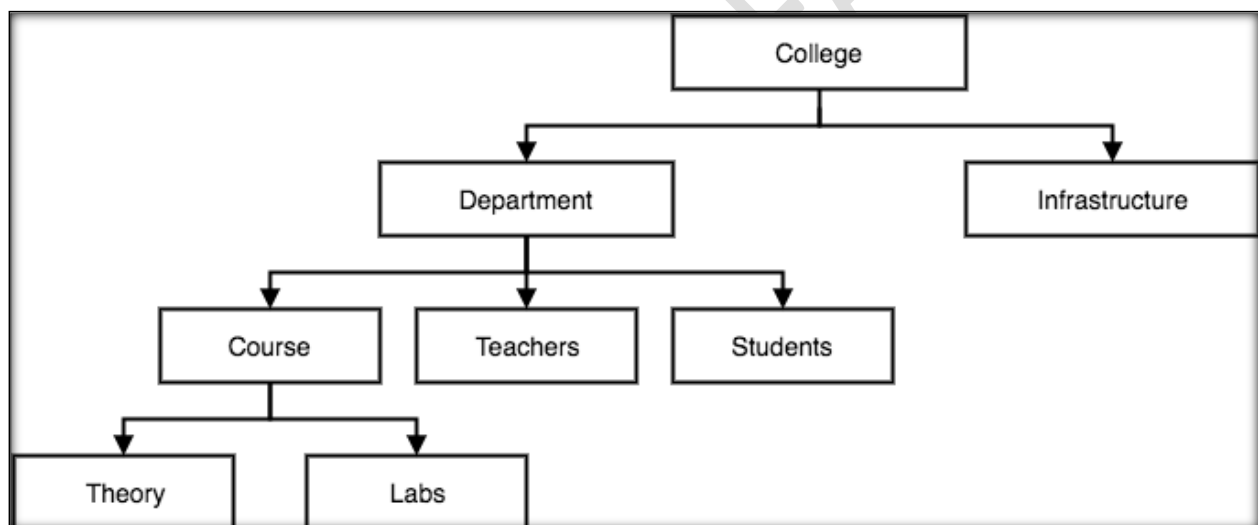
- Flat-file Data Model
- Hierarchical Data Model
- Network Data Model
- Relational Data Model

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.

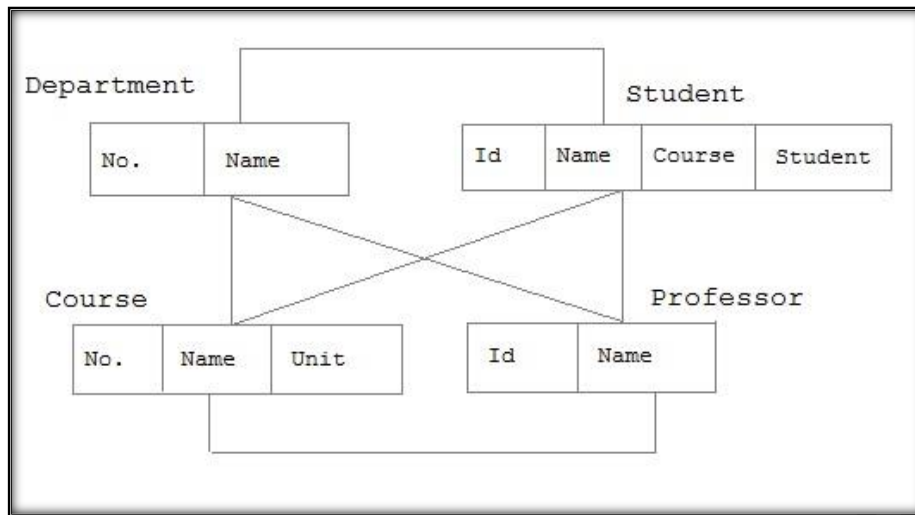
Hierarchical Data Model:

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

- 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

Diagram illustrating the structure of the STUDENT relation:

- Relation Name:** STUDENT
- Attributes:** Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa
- Tuples:** Benjamin Bayer, Chung-cha Kim, Dick Davidson, Rohan Panchal, Barbara Benson

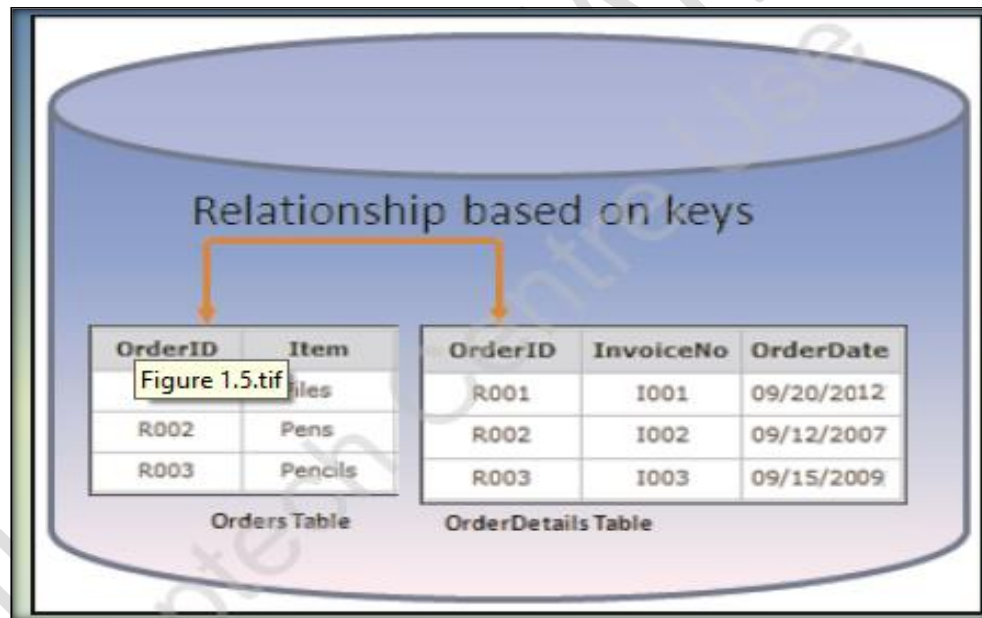
Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492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



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 TOP Clause:

Syntax:

`select top 3 * from employee`

SQL PERCENTAGE:

`select top 50 percent * from employee`

SQL IS NULL:

`select * from employee where did is null`

SQL IS NOT NULL:

`select * from employee where email is not null`

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 CONSTRAINTS:

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 CONSTRAINTS:

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 CONSTRAINTS:

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 CONSTRAINTS:

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 CONSTRAINTS:

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

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)  
);
```

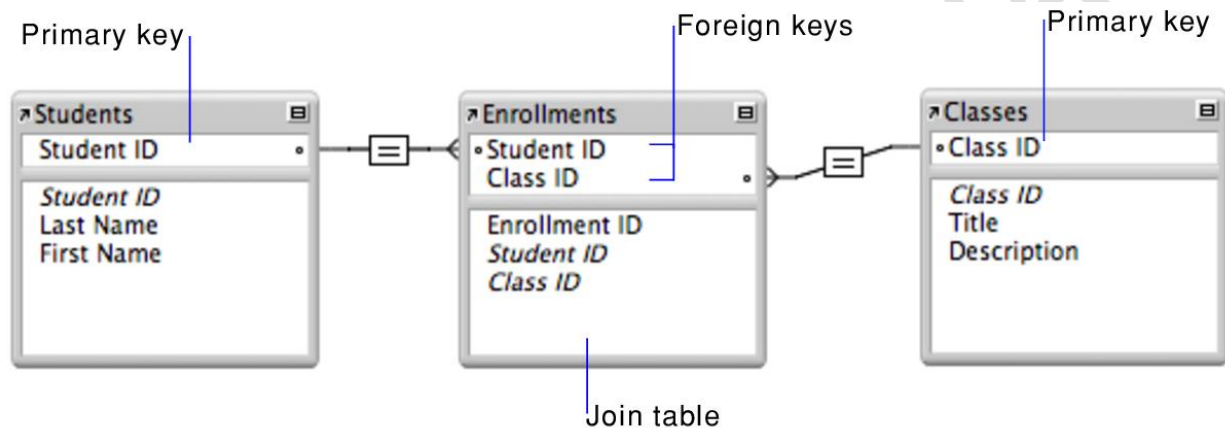
```
CREATE TABLE Orders(  
  O_ID INT primary key,  
  O_NAME VARCHAR (20) NOT NULL,  
  C_ID int, foreign key (C_ID) references CUSTOMERS(C_ID)  
);
```

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:

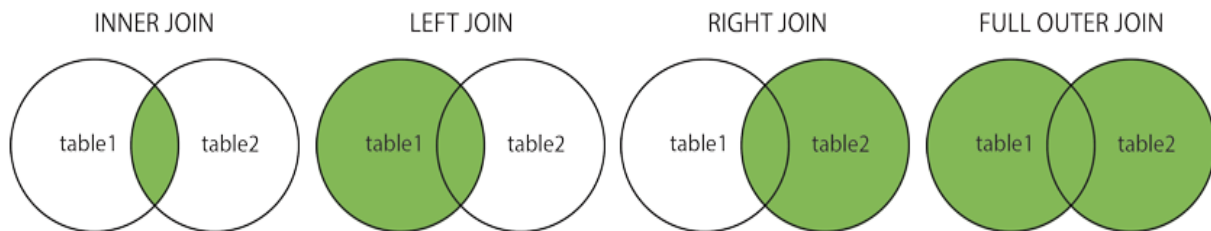
```
SELECT column_name(s)
FROM table_name AS alias_name;
```

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

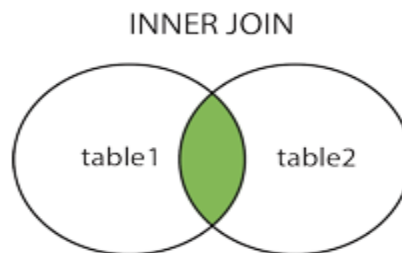


INNER JOINS:

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;
```



General Example:

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

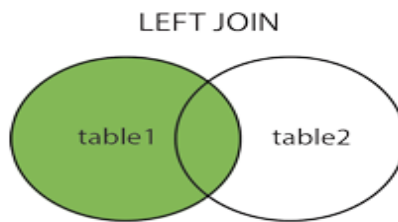
Join Three Tables:

```
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:

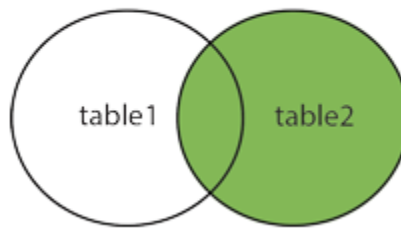
```
SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;
```

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).

RIGHT JOIN



Syntax:

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

Example:

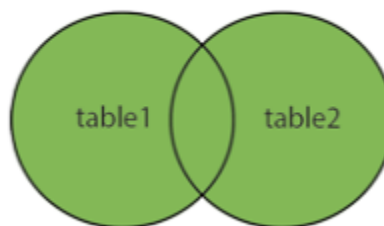
```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders RIGHT JOIN Employees ON Orders.EmployeeID =  
Employees.EmployeeID ORDER BY Orders.OrderID;
```

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

FULL OUTER JOINS:

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

CROSSJOIN



Syntax:

```
SELECT column_name(s) FROM table1  
full outer JOIN table2 on table.id = table2.id;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID FROM Customers  
FULL OUTER JOIN Orders on Customer.orderid = Orders.orderid;
```

SELF JOIN:

A self-join is a regular join, but the table is joined with itself.

Example:

EMP ID	EMP NAME	MANAGER ID
1	Rehman	4
2	Amir	5
3	Anas	4
4	Hamza	5
5	Hammad	4

```
Select A.emp_name as EMPLOYEE, B.Emp_Name as MANAGER from  
Employee_Manager as A
```

```
Inner join Employee_Manager as B on A.Manager_ID = B. EMP_ID
```

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.

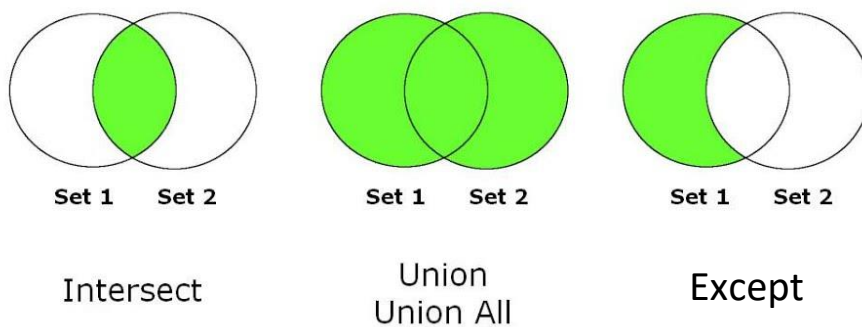
Syntax:

```
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:

The UNION operator selects only distinct values by default. To allow duplicate values, use **UNION ALL**.

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;
```

SQL INTERSECT:

SQL INTERSECT operator combines two select statements and returns only the dataset that is common in both the statements.

Example:

```
SELECT City FROM Customers  
INTERSECT  
SELECT City FROM Suppliers;
```

SQL EXCEPT:

The SQL **EXCEPT** clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means EXCEPT returns only rows, which are not available in the second SELECT statement.

Example:

```
SELECT City FROM Customers  
EXCEPT  
SELECT City FROM Suppliers 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 esalary = (SELECT MIN(esalary) FROM  
employee);
```

Another Example:

Suppose we want details of customers who have placed an order. Here's how we can do that using a subquery:

```
SELECT customer_id, first_name FROM Customers WHERE customer_id IN (SELECT customer_id FROM Orders);
```

Same SQL QUERY using Join:

In some scenarios, we can get the same result set using a join

Example:

```
SELECT DISTINCT Customers.customer_id, Customers.first_name  
FROM Customers  
INNER JOIN Orders  
ON Customers.customer_id = Orders.customer_id  
ORDER BY Customers.customer_id;
```

TYPES OF SUBQUERIES IN SQL:

1. Scalar Sub Queries.
2. Multivalued Sub Queries

1) Scalar Sub Queries:

Sub queries that return one row to the outer SQL statement.

Operators: = > >= < <= !=

2) Multivalued Sub Queries:

Sub queries that return more than one row to the outer SQL Statement

Operators: in , any , all

Example:

```
SELECT * FROM Employee WHERE salary < any (SELECT salary FROM Employee  
where name = 'ali' or name = 'afan');
```


Another Example:

SELECT * **FROM** Employee **WHERE** salary < all (**SELECT** salary **FROM** Employee where name = 'ali' or name = 'afan');

There are further divided into two categories:

SELF Contained Sub Queries:

- These queries are written as standalone queries, without any dependencies on the outer query.
- A self-contained subquery is processed once when the outer query runs and passes its results to the outer query.

CO Related Sub Queries:

- These queries reference one or more columns from the outer query and therefore, depend on the outer query.
- Co related sub queries cannot be run separately from the outer query.

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');

SQL Alter Statement:

The ALTER TABLE statement is used to Modify Database Name.

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 Database Change Database Name:

Example:

```
alter database batch11g modify name = batch11g2
```

Alter Table Change Table Name:

```
execute sp_rename 'student', 'student_info ';
```

Alter Table Add Column:

Example:

```
alter table student_info add city varchar(50)
```

Alter Table Drop Column:

Example:

```
ALTER TABLE student drop column city
```

Alter Table change Data type of column:

```
alter table student_info alter column salary int
```

Alter Table Add constraints on column:

```
ALTER TABLE student_info alter column city varchar(40) not null;
```

Add Another Constraints on Column:

```
alter table student_info add unique(email)
```

Alter Table Drop Constraint:

```
alter table student_info drop constraint UQ__student__3213E83E3C16B3FA
```

Alter Table Modify Column constraint:

Syntax:

```
ALTER TABLE table_name  
ADD UNIQUE(email) ;
```

Alter Table Add Foreign Key in Existing Column:

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

Example:

```
alter table student_info add foreign key(tid) references teacher(tid)
```

Alter Table ADD Default Constraint on Table:

Alter Table customer add default 18 for cus_age.

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,sname,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
<u>CEILING</u>	Returns the smallest integer value that is >= to a number
FLOOR	Returns the largest integer value that is <= to a number
ROUND	Rounds a number to a specified number of decimal places
/	Use division number

%	Returns the remainder of a number divided by another number
PI	Returns the value of PI
POWER	Returns the value of a number raised to the power of another number
RAND	Returns a random number
SQRT	Returns the square root of a number

MySQL String Functions:

Function	Description
LEN	Returns the length of a string (in bytes)
CONCAT	Adds two or more expressions together
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
UPPER	Converts a string to upper-case

REVERSE()	Reverse all the characters in the given string
------------------	--

DATE & TIME FUNCTION:

Function	Description
GetDate()	Returns the current date and time
Sysdatetime()	Returns the current date and time with 4 precession
CURRENT_TIMESTAMP	Returns the current date and time
DATENAME(MONTH,'2022-03-02')	Returns the year , month , day , minute , hour , seconds
DateDiff(interval , date1 , date 2)	Returns the difference between two dates
Date Add(day , 4 , getdate())	Add or subtracts a specified time interval from a date
DatePart(year , getdate())	Returns a single part of a date / time
DAY	Returns the day of the month for a given date
MONTH	Returns the month part for a given date
YEAR	Returns the year part for a given date
IsDate	Check if the expression is a valid date.

User Defined Function:

User Defined Function is the code that extends the functionality of SQL server by adding external code can work same as inbuilt functions like concat(), length() in SQL. 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.

There are three types of user-defined functions in SQL Server:

1. Scalar Functions
2. Inline Table values functions
3. Multi-Statement Table Valued Functions.

Scalar Functions:

SQL Server scalar function takes one or more parameters and returns a single value.

The returned value can be any data type, except : text, ntext timestamp.

Example:

```
create function showmsg()  
returns varchar(50)  
as  
begin  
    return 'welcome this is good message'  
end
```

Calling Function:

```
select dbo.showmsg()
```

Parameterized Functions:

```
create function takenumber(@num as int)
returns int
as
begin
    return (@num + 10)
end
```

Calling Function:

```
select dbo.takenumber(20)
```

Drop Function:

```
drop function dbo.takenumber;
```

Inline Table values Functions:

Contains a single TSQL statement and returns a Table Set.

It return table instead of like int, varchar etc.

Example:

```
create function getemployee()
returns table
as
return (select * from employee)
```

Example:

```
select * from getemployee()
```


Multi-Statement Table Valued Functions:

- A multi-statement table-valued function is a table-valued function that returns the result of multi statements.

Example:

```
create function getempbysalary( @salary varchar(50))
returns @mytable table (empid int,empname varchar(50),salary int)
as
begin
    insert into @mytable

    select eid,ename,esalary from employee where esalary < @salary
    return

end
```

Calling Function:

```
select * from dbo.getempby@salary(24000)
```

SQL Queries:

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.

These SQL commands which can be categorized into five categories 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, REVOKE

TCL (Transaction Control Language):

Transactions group a set of tasks into a single execution unit. 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.

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.

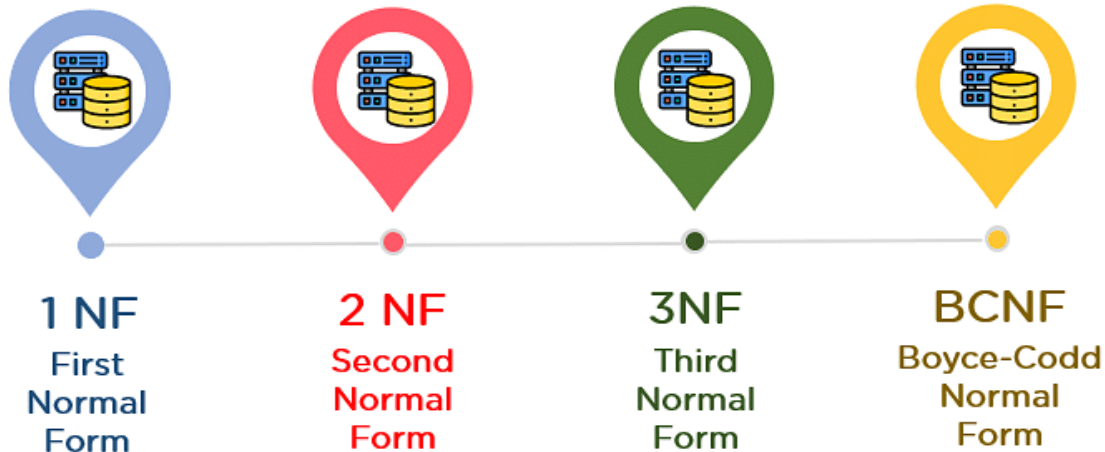
- one-to-one
- one-to-many
- many-to-many

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

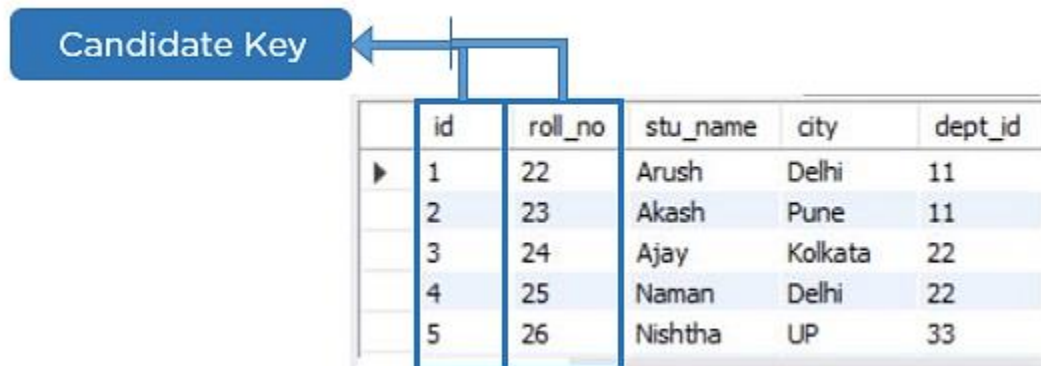
By applying the First Normal Form, you achieve atomicity, and also every column has unique values.

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.

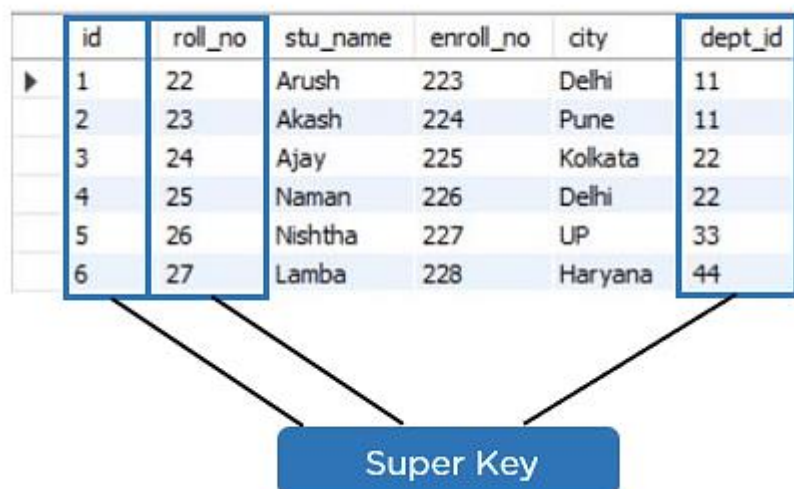


	id	roll_no	stu_name	city	dept_id
▶	1	22	Arush	Delhi	11
	2	23	Akash	Pune	11
	3	24	Ajay	Kolkata	22
	4	25	Naman	Delhi	22
	5	26	Nishtha	UP	33

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 $A \rightarrow C$ (A determines C) indirectly, because of $A \rightarrow B$ and $B \rightarrow C$ (where it is not the case that $B \rightarrow A$).
- 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 Codd Normal 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 $X \rightarrow Y$. 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 \rightarrow 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.

SQL 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.

View can be used as a mechanism to implement Row and Column level security.

A view is created with the CREATE VIEW statement.

Example:

```
create view vw_getemp
as
select A.eid,A.ename,A.esalary,B.dname from employee as A inner join
department as B on A.did = B.did;
```

SQL VIEW CODE GET:

```
sp_helptext vw_getemp
```

DROP VIEW:

Example:

```
drop view vw_getemp;
```

ALTER VIEW STATEMENT:

```
alter view vw_getemp  
as  
select A.eid,A.ename,B.dname from employee as A inner join  
department as B on A.did = B.did;
```

Main Point on View or Disadvantage:

Although there are many advantages to views, the main disadvantage to using views rather than real tables is **performance degradation**. Because views only create the appearance of a table, not a real table, the query processor must translate queries against the view into queries against the underlying source tables.

SQL CREATE INDEX STATEMENT:

- To facilitate quick retrieval of data from a database.
- An Index Allow to find specific data without scanning through the entire table.
- Index are created on tables and views.
- Example of index is Book.

Sequence find 100:

1	2	3	4	5	6	100
---	---	---	---	---	---	-------	-----

Example:

```
create index ix_empsalary on  
employee (esalary ASC)
```

Drop Index:

Syntax:

Drop index tablename.indexname;

Example:

drop index employee.ix_empname

Create Unique Index Syntax:

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

Drop Index Statement:

DROP INDEX tbl_name. *index_name*;

Get All Table Index:

sp_helpindex employee

CLUSTERED INDEX:

- A clustered index cause records to be physically stored in a sorted or sequential order.
- A Clustered index determines the actual order in which data is stored in the database. Hence, you can create only one clustered index in a table.
- Uniqueness of a value in a clustered index is maintained explicitly using the unique keyword or implicit using an internal unique identifier.
- Clustered Index is a same as dictionary where the data is arranged by alphabetical order.

Example:

create clustered index ix_empid on
employee (id)

NON-CLUSTERED INDEX:

- A Non-Clustered Index is as same as to an index of a book.
- The data is stored in one place, and index is stored in another place.
- Since, The Non-Clustered Index is stored separately from the actual data, A Table can have more than one non-clustered index.
- Just like how a book can have index by chapters at the beginning and another index by common terms at the end.

Example:

`create nonclustered index ix_empsalary on
employee (esalary ASC)`

Unique and Non Unique Index:

- A unique index can be created on a column that does not have any duplicate values.
- Once a unique index is created, duplicate values will not be accepted in the column.
- Thus, unique indexes should be created only on columns where uniqueness of values is a key characteristic. A unique index ensures entity integrity in a table.
- If a table definition has a PRIMARY KEY or a column with a UNIQUE constraint, SQL SERVER automatically creates a unique index when you execute the CREATE Table statement.
- Uniqueness is a property of an index, and both clustered and non-clustered indexes can be unique.

SQL STORED PROCEDURE:

A stored procedure is a set of structured query language (SQL) statements with an assigned name, which are stored in a relational database management system as a group, so it can be reused and shared by multiple programs.

Types of Stored Procedure:

1. System Stored Procedure
2. User Defined Stored Procedure

Example:

```
create procedure sp_getemp  
as  
begin  
    select * from employee;  
end
```

Call Procedure:

```
sp_getemp
```

OR

```
execute sp_getemp
```

Parametrized Procedure:

```
create procedure sp_getempbyid  
@id int  
as  
begin  
    select * from employee where eid = @id;  
end
```

Call Parametrized Procedure:

```
sp_getempbyid 3 OR execute sp_getempbyid 3
```

Show Procedure Code:

```
sp_helptext sp_getempbyid
```

Drop Procedure:

```
drop procedure sp_getemp
```

Procedure with Encryption:

```
create procedure sp_getempbyid  
@id int  
with encryption  
as  
begin  
    select * from employee where eid = @id;  
end
```

Procedure with Input & Output Parameters:

Example:

```
create procedure countemp
@city varchar(40),
@counte int output
as
begin
    select @counte = count(id) from emp2 where city = @city;
end
```

Calling Procedure:

```
declare @totalemp int

execute countemp 'karachi' , @totalemp output

select @totalemp
```

Difference Between Functions and Stored Procedures:

Functions	Stored Procedures
Functions must return a value	It may or not return values.
It can have only input parameters	It can have both input and output parameters
Function can be called from store procedures	Stored Procedures cannot be called from a function
Function allow only select statement in it	It allow select as well as DML statement init
We cannot use transaction in function.	We can use transaction in stored procedure
We use select command execute a function	We use exec or execute to execute procedure

SQL TRIGGERS:

A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database server.

There are Three types of triggers.

1. DML Triggers (Data manipulation Language) Insert, Update, Delete

2. DDL Triggers (Data Definition Language) Create, Alter
3. Logon Triggers

DML Triggers can be of 2 types:

1. After Trigger (Also called for triggers)
2. Instead of triggers

DML Trigger:

Example:

```
create trigger tr_employee for insert
on employee after insert
as
begin
    print 'Something Insert into Employee';
end
```

Another Example of Trigger with Magical Table:

INSERTED and DELETED Keywords in MySQL:

- SQL Triggers provide us 2 magical / virtual tables called **INSERTED** and **DELETED**.
- When we insert a row in a table then that row is also inserted in **INSERTED** table.
- When we delete a row from a table then that row is also **DELETED** in OLD table.

```
alter trigger tr_employee for insert
on employee after insert
as
begin
    select * from inserted;
end
```

Example Trigger for Auditing:

```

create trigger tr_employeefordelete
on employee after delete
as
begin
    declare @id int
    select @id = eid from deleted

    insert into empaudit values('Employee ID : ' +
    cast(@id as varchar(40)) + ' is Deleted at : ' + cast(GETDATE() as
varchar(50)))
end

```

DML Instead of Triggers:

Example:

```

create trigger tr_employeeinsert
on employee
instead of insert
as
begin
    print 'not allow to insert in employee table';
end

```

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:

```

begin transaction
update employee set ename = 'farhan ali rehman' where eid = 1
commit transaction

```

Another Example:

```

begin transaction
update employee set ename = 'muhammad farhan' where eid = 1
rollback transaction

```

Definition Transaction:

A logical unit of work must exhibit four properties, called the **atomicity**, **consistence**, **isolation**, and **durability** (**ACID**) properties.

Properties of Transaction in MySQL:

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

- **Atomicity** – If the transaction has many operations then all should be committed. It means ALL or None. It manages by Transaction Manager.
- **Consistency** – This ensures that the database properly changes states upon a successfully committed transaction.

Example: A = 2000, B = 3000 A+B = 5000 --before transaction

A = 2000 – 1000 = 1000

B = 3000 + 1000 = 4000

A + B = 5000

--after transaction

- **Isolation** – The Operation that are performed must be isolated from the other operations on the same server on the same database. It means each transaction must be executed without knowing what is happening to other transactions.
- **Durability** – The Operation that are performed on the database must be saved and stored in the database permanently.

In SQL, the transactions begin with the statement **BEGIN Transaction** 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.

