SHANMUGANATHAN ENGINEERING COLLEGE
ARASAMPATTI – 622 507
Accredited by NAAC, Approved by AICTE,
An ISO 9001:2015 Certified Institution

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

Subject Code: CS3481
Subject Name: Database Management Systems Lab
Program Name: B. E., CSE
Year & Sem: II & IV
Regulation: 2021

CS3481 DATABASE MANAGEMENT SYSTEMS LABORATORY L T P C
0 0 3 1.5

COURSE OBJECTIVES:

- To learn and implement important commands in SQL.
- To learn the usage of nested and joint queries.
- To understand functions, procedures and procedural extensions of databases.
- To understand design and implementation of typical database applications.
- To be familiar with the use of a front end tool for GUI based application development.

LIST OF EXPERIMENTS:

1. Create a database table, add constraints (primary key, unique, check, Not null), insert rows, Update and delete rows using SQL DDL and DML commands.
2. Create a set of tables, add foreign key constraints and incorporate referential integrity.
3. Query the database tables using different 'where' clause conditions and also implement Aggregate functions.
4. Query the database tables and explore sub queries and simple join operations.
5. Query the database tables and explore natural, equi and outer joins.
6. Write user defined functions and stored procedures in SQL.
7. Execute complex transactions and realize DCL and TCL commands.
8. Write SQL Triggers for insert, delete, and update operations in a database table.
9. Create View and index for database tables with a large number of records.
10. Create an XML database and validate it using XML schema.
11. Create Document, column and graph based data using NOSQL database tools.
12. Develop a simple GUI based database application and incorporate all the above mentioned Features
13. Case Study using any of the real life database applications from the following list

a) Inventory Management for a EMart Grocery Shop

b) Society Financial Management

c) Cop Friendly App – Eseva

d) Property Management – eMall

e) Star Small and Medium Banking and Finance

● Build Entity Model diagram. The diagram should align with the business and functional Goals stated in the application.

Apply Normalization rules in designing the tables in scope.
● Prepared applicable views, triggers (for auditing purposes), functions for enabling Enterprise grade features.
● Build PL SQL / Stored Procedures for Complex Functionalities,
Ex EOD Batch Processing for calculating the EMI for Gold Loan for each eligible Customer.
● Ability to showcase ACID Properties with sample queries with appropriate settings

List of Equipments :(30 Students per Batch)

MYSQL / SQL: 30 Users

TOTAL: 45 PERIODS

COURSE OUTCOMES:
At the end of this course, the students will be able to:
CO1: Create databases with different types of key constraints.
CO2: Construct simple and complex SQL queries using DML and DCL commands.
CO3: Use advanced features such as stored procedures and triggers and incorporate in GUI based application development.
CO4: Create an XML database and validate with meta-data (XML schema).
CO5: Create and manipulate data using NOSQL database.

Ex No: 1	**Create a database table, add constraints (primary key, unique, check, Not null), Insert rows, Update and delete rows using SQL DDL and DML commands.**

Aim:

To create table and Execute Data Definition Commands, Data Manipulation Commands for Inserting, Deleting, Updating and Retrieving Tables with constraints.

SQL: create command

Create is a DDL SQL command used to create a table or a database in relational database management system.

Creating a Database

To create a database in RDBMS, create command is used. Following is the syntax,

CREATE DATABASE <DB_NAME>;

Example for creating Database

CREATE DATABASE Test;

The above command will create a database named Test, which will be an empty schema without any table.

To create tables in this newly created database, we can again use the create command.

Creating a Table

Create command can also be used to create tables. Now when we create a table, we have to specify the details of the columns of the tables too. We can specify the names and data types of various columns in the create command itself.

Following is the syntax,

CREATE TABLE <TABLE_NAME>

(

column_name1 datatype1,

column_name2 datatype2,

column_name3 datatype3,

column_name4 datatype4

);

Create table command will tell the database system to create a new table with the given table name and column information.

Most commonly used data types for Table columns

Here we have listed some of the most commonly used data types used for columns in tables.

Datatype	**Use**
INT	Used for columns which will store integer values.
FLOAT	Used for columns which will store float values.
DOUBLE	Used for columns which will store float values.
VARCHAR	Used for columns which will be used to store characters and integers, basically a string.
CHAR	Used for columns which will store char values (single character).
DATE	Used for columns which will store date values.
TEXT	Used for columns which will store text which is generally long in length. For example, if you create a table for storing profile information of a social networking website, then for **about me** section you can have a column of type TEXT.

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 are mainly categorized into five categories as:

1. **DDL – Data Definition Language**
2. **DQL – Data Query Language**
3. **DML – Data Manipulation Language**
4. **DCL – Data Control Language**
5. **TCL – Transaction Control Language**

DDL (Data Definition Language):

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema.

It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

DDL is a set of SQL commands used to create, modify, and delete database structures but not data.

These commands are normally not used by a general user, who should be accessing the database via an application.

List of DDL commands:

- **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- **DROP**: This command is used to delete objects from the database.
- **ALTER:** This is used to alter the structure of the database.
- **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT**: This is used to add comments to the data dictionary.
- **RENAME:** This is used to rename an object existing in the database.

DQL (Data Query Language):

DQL statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it.

It includes the SELECT statement. This command allows getting the data out of the database to perform operations with it.

List of DQL:

- **SELECT:** It is used to retrieve data from the database.

DML (Data Manipulation Language):

The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

List of DML commands:

- **INSERT** : It is used to insert data into a table.
- **UPDATE:** It is used to update existing data within a table.
- **DELETE** : It is used to delete records from a database table.
- **LOCK:** Table control concurrency.
- **CALL:** Call a PL/SQL or JAVA subprogram.
- **EXPLAIN PLAN:** It describes the access path to data.

DCL (Data Control Language):

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

- **GRANT:** This command gives users access privileges to the database.
- **REVOKE:** This command withdraws the user's access privileges given by using the GRANT command.

TCL (Transaction Control Language):

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure. You can explore more about transactions here. Hence, the following TCL commands are used to control the execution of a transaction:

- **COMMIT:** Commits a Transaction.
- **ROLLBACK:** Rollbacks a transaction in case of any error occurs.

- **SAVEPOINT:** Sets a save point within a transaction.
- **SET TRANSACTION:** Specifies characteristics for the transaction.

DDL (Data Definition Language):

Examples of CREATE Command in SQL

Example 1: This example describes how to create a new database using the CREATE DDL command.

Syntax to Create a Database:

CREATE Database Database_Name;

Suppose, you want to create a Books database in the SQL database. To do this, you have to write the following DDL Command:

Create Database Books;

Example 2: This example describes how to create a new table using the CREATE DDL command.

Syntax to create a new table:

Suppose, you want to create a **Student** table with five columns in the SQL database. To do this, you have to write the following DDL command:

CREATE TABLE Student

(Roll_No. Int , First_Name Varchar (20) , Last_Name Varchar (20) , Age Int , Marks Int) ;

Examples of DROP Command in SQL

Example 1: This example describes how to remove the existing table from the SQL database.

Syntax to remove a table:

DROP TABLE Table_Name;

Suppose, you want to delete the Student table from the SQL database. To do this, you have to write the following DDL command:

DROP TABLE Student;

Examples of ALTER Command in SQL

Example 1: This example shows how to add a new field to the existing table.

Syntax to add a Newfield in the table:

ALTER TABLE name_of_table **ADD** column_name column_definition;

Suppose, you want to add the 'Father's_Name' column in the existing Student table. To do this, you have to write the following DDL command:

ALTER TABLE Student **ADD** Father's_Name **Varchar**(60);

Example 2: This example describes how to remove the existing column from the table.

Syntax to remove a column from the table:

ALTER TABLE name_of_table **DROP** Column_Name_1 , column_Name_2 , ….., co
lumn_Name_N;

Suppose, you want to remove the Age and Marks column from the existing Student table. To do this, you have to write the following DDL command:

ALTER TABLE StudentDROP Age, Marks;

Example 3: This example describes how to modify the existing column of the existing table.

Syntax to modify the column of the table:

ALTER TABLE table_name **MODIFY** (column_name column_datatype(**size**));

Suppose, you want to change the character size of the Last_Namefield of the Student table. To do this, you have to write the following DDL command:

ALTER TABLE table_name **MODIFY** (Last_Name **varchar**(25));

Syntax of TRUNCATE command

1. **TRUNCATE TABLE** Table_Name;

Example

Suppose, you want to delete the record of the Student table. To do this, you have to write the following TRUNCATE DDL command:

1. **TRUNCATE TABLE** Student;

The above query successfully removed all the records from the student table. Let's verify it by using the following SELECT statement:

1. **SELECT** * **FROM** Student;

Syntax of RENAME command

1. RENAME **TABLE** Old_Table_Name **TO** New_Table_Name;

Example

1. RENAME **TABLE** Student **TO** Student_Details ;

This query changes the name of the table from Student to Student_Details.

DML(Data Manipulation Language):

Syntax of INSERT Command

INSERT INTO TABLE_NAME (column_Name1 , column_Name2 , column_Name3 , ... column_NameN) VALUES (value_1, value_2, value_3, ... value_N) ;

Examples of INSERT Command

Example 1: This example describes how to insert the record in the database table.

Let's take the following student table, which consists of only 2 records of the student.

Stu_Id	Stu_Name	Stu_Marks	Stu_Age
101	Ramesh	92	20
201	Jatin	83	19

Suppose, you want to insert a new record into the student table. For this, you have to write the following DML INSERT command:

INSERT INTO Student (Stu_id, Stu_Name, Stu_Marks, Stu_Age) VALUES (104, Anmol, 89, 19);

Syntax of SELECT DML command

SELECT column_Name_1, column_Name_2, ….., column_Name_N **FROM** Name_of_table;

Here, **column_Name_1, column_Name_2, ….., column_Name_N** are the names of those columns whose data we want to retrieve from the table.

If we want to retrieve the data from all the columns of the table, we have to use the following SELECT command:

SELECT * **FROM** table_name;

Examples of SELECT Command

Example 1: This example shows all the values of every column from the table.

SELECT * **FROM** Student;

This SQL statement displays the following values of the student table:

Student ID	Student Name	Student Marks
BCA1001	Ahoy	85
BCA1002	Annul	75
BCA1003	Bheem	60

BCA1004	Ram	79
BCA1005	Sumit	80

Example 2: This example shows all the values of a specific column from the table.

SELECT Emp_Id, Emp_Salary **FROM** Employee;

This SELECT statement displays all the values of **Emp_Salary** and **Emp_Id** column of **Employee** table:

Emp_Id	Emp_Salary
201	25000
202	45000
203	30000
204	29000
205	40000

Example 3: This example describes how to use the WHERE clause with the SELECT DML command.

Let's take the following Student table:

Student_ID	Student_Name	Student_Marks
BCA1001	Abhay	80
BCA1002	Ankit	75
BCA1003	Bheem	80
BCA1004	Ram	79
BCA1005	Sumit	80

If you want to access all the records of those students whose marks is 80 from the above table, then you have to write the following DML command in SQL:

1. **SELECT** * **FROM** Student **WHERE** Stu_Marks = 80;

The above SQL query shows the following table in result:

Student_ID	**Student_Name**	**Student_Marks**
BCA1001	Abhay	80
BCA1003	Bheem	80
BCA1005	Sumit	80

Syntax of UPDATE Command

UPDATE Table_name **SET** [column_name1= value_1, ….., column_nameN = value_N] **WH ERE** CONDITION;

Here, 'UPDATE', 'SET', and 'WHERE' the SQL keywords, and 'Table name' are is the name of the table whose values you want to update.

Examples of the UPDATE command

Example 1: This example describes how to update the value of a single field.

Let's take a Product table consisting of the following records:

Product_Id	**Product_Name**	**Product_Price**	**Product_Quantity**
P101	Chips	20	20
P102	Chocolates	60	40
P103	Maggi	75	5
P201	Biscuits	80	20
P203	Namkeen	40	50

Suppose, you want to update the Product_Price of the product whose Product_Id is P102. To do this, you have to write the following DML UPDATE command:

UPDATE Product **SET** Product_Price = 80 **WHERE** Product_Id = 'P102' ;

Example 2: This example describes how to update the value of multiple fields of the database table.

UPDATE Student **SET** Stu_Marks = 80, Stu_Age = 21 **WHERE** Stu_Id = 103 AND S tu_Id = 202;

Syntax of DELETE Command

DELETE FROM Table_Name **WHERE** condition;

Examples of DELETE Command

Example 1: This example describes how to delete a single record from the table.

Let's take a Product table consisting of the following records:

Product_Id	Product_Name	Product_Price	Product_Quantity
P101	Chips	20	20
P102	Chocolates	60	40
P103	Maggi	75	5
P201	Biscuits	80	20
P203	Namkeen	40	50

Suppose, you want to delete that product from the Product table whose Product_Id is P203. To do this, you have to write the following DML DELETE command:

1. **DELETE FROM** Product **WHERE** Product_Id = 'P202' ;

Example 2: This example describes how to delete the multiple records or rows from the database table.

DELETE FROM Student **WHERE** Stu_Marks > 70 ;

SQL CASE

The **CASE** is a statement that operates if-then-else type of logical queries. This statement returns the value when the specified condition evaluates to True. When no condition evaluates to True, it returns the value of the ELSE part.

In Structured Query Language, CASE statement is used in SELECT, INSERT, and DELETE statements with the following three clauses:

1. **WHERE Clause**
2. **ORDER BY Clause**
3. **GROUP BY Clause**

This statement in SQL is always followed by at least one pair of WHEN and THEN statements and always finished with the END keyword.

The CASE statement is of two types in relational databases:

1. Simple CASE statement
2. Searched CASE statement

Syntax of CASE statement in SQL

```
CASE <expression>

WHEN condition_1 THEN statement_1

WHEN condition_2 THEN statement_2 …….

WHEN condition_N THEN statement_N

ELSE result

END;
```

- Here, the CASE statement evaluates each condition one by one.
- If the expression matches the condition of the first WHEN clause, it skips all the further WHEN and THEN conditions and returns the statement_1 in the result.
- If the expression does not match the first WHEN condition, it compares with the seconds WHEN condition. This process of matching will continue until the expression is matched with any WHEN condition.
- If no condition is matched with the expression, the control automatically goes to the ELSE part and returns its result. In the CASE syntax, the ELSE part is optional.
- In Syntax, CASE and END are the most important keywords which show the beginning and closing of the CASE statement.

<u>Examples of CASE statement in SQL</u>

Let's take the Student_Details table, which contains roll_no, name, marks, subject, and city of students.

Roll_No	Stu_Name	Stu_Subject	Stu_Marks	Stu_City
2001	Akshay	Science	92	Noida
2002	Ram	Math	49	Jaipur
2004	Shyam	English	52	Gurgaon
2005	Yatin	Hindi	45	Lucknow
2006	Manoj	Computer	70	Ghaziabad
2007	Sheetal	Math	82	Noida
2008	Parul	Science	62	Gurgaon

Example 1: The following SQL statement uses single WHEN and THEN condition to the CASE statement:

SELECT Roll_No, Stu_Name, Stu_Subject, Stu_marks,

CASE

WHEN Stu_Marks >= 50 THEN 'Student_Passed'

ELSE 'Student_Failed'

END AS Student_Result

FROM Student_Details;

Explanation of above query:

Here, the CASE statement checks that if the **Stu_Marks** is greater than and equals 50, it returns **Student_Passed** otherwise moves to the **ELSE** part and returns **Student_Failed** in the **Student_Result** column.

Output:

Roll_No	Stu_Name	Stu_Subject	Stu_Marks	Student_Result
2001	Akshay	Science	92	Student_Passed
2002	Ram	Math	49	Student_Failed
2004	Shyam	English	52	Student_Passed
2005	Yatin	Hindi	45	Student_Failed
2006	Manoj	Computer	70	Student_Passed
2007	Sheetal	Math	82	Student_Passed
2008	Parul	Science	62	Student_Passed

Example 2: The following SQL statement adds more than one WHEN and THEN condition to the CASE statement:

SELECT Roll_No, Stu_Name, Stu_Subject, Stu_marks,

CASE

WHEN Stu_Marks >= 90 THEN 'Outstanding'

WHEN Stu_Marks >= 80 AND Stu_Marks < 90 THEN 'Excellent'

WHEN Stu_Marks >= 70 AND Stu_Marks < 80 THEN 'Good'

WHEN Stu_Marks >= 60 AND Stu_Marks < 70 THEN 'Average'

WHEN Stu_Marks >= 50 AND Stu_Marks < 60 THEN 'Bad'

WHEN Stu_Marks < 50 THEN 'Failed'

END AS Stu_Remarks

FROM Student_Details;

Example 3:

Let's take another Employee_Details table which contains Emp_ID, Emp_Name, Emp_Dept, and Emp_Salary.

Emp_Id	Emp_Name	Emp_Dept	Emp_Salary
1	Akshay	Finance	9000
2	Ram	Marketing	4000
3	Shyam	Sales	5000
4	Yatin	Coding	4000
5	Manoj	Marketing	5000
1	Akshay	Finance	8000
2	Ram	Coding	6000
3	Shyam	Coding	4000
4	Yatin	Marketing	8000
5	Manoj	Finance	3000

The following SQL query uses GROUP BY clause with CASE statement:

SELECT Emp_Id, Emp_Name, Emp_Dept, sum(Emp_Salary) as Total_Salary,

CASE

WHEN SUM(Emp_Salary) >= 10000 THEN 'Increment'

ELSE 'Constant'

END AS Emp_Remarks

FROM Employee_Details

GROUP BY Emp_id, Emp_Name;

Output:

Emp_Id	Emp_Name	Emp_Dept	Total_Salary	Emp_Remarks
1	Akshay	Finance	17000	Increment
2	Ram	Marketing	9000	Decrement
3	Shyam	Sales	10000	Increment
4	Yatin	Coding	12000	Increment
5	Manoj	Marketing	8000	Decrement

Example 4: In this example, we use the ORDER BY clause with a CASE statement in SQL:

Let's take another Employee_Details table which contains Emp_ID, Emp_Name, Emp_Dept, and Emp_Age.

We can check the data of Employee_Details by using the following query in SQL:

Select * **From** Employee_Details;

Output:

Emp_Id	Emp_Name	Emp_Dept	Emp_Age
1	Akshay	Finance	23
2	Ram	Marketing	24
3	Balram	Sales	25
4	Yatin	Coding	22
5	Manoj	Marketing	23
6	Sheetal	Finance	24
7	Parul	Finance	22
8	Yogesh	Coding	25

9	Naveen	Marketing	22
10	Tarun	Finance	23

The following SQL query shows all the details of employees in the ascending order of employee names:

SELECT * FROM Employee_Details

ORDER BY Emp_Name;

Output:

Emp_Id	Emp_Name	Emp_Dept	Emp_Age
1	Akshay	Finance	23
3	Balram	Sales	25
5	Manoj	Marketing	23
9	Naveen	Marketing	22
7	Parul	Finance	22
2	Ram	Marketing	24
6	Sheetal	Finance	24
10	Tarun	Finance	23
4	Yatin	Coding	22
8	Yogesh	Coding	25

If you want to show those employees at the top who work in the Coding Department, then for this operation, you have to use single WHEN and THEN statement in the CASE statement as shown in the following query:

```
SELECT * FROM Employee_Details
ORDER BY CASE WHEN Emp_Dept = 'Coding' THEN 0
ELSE 1 END, Emp_Name;
```

Output:

Emp_Id	Emp_Name	Emp_Dept	Emp_Age
4	Yatin	Coding	22
8	Yogesh	Coding	25
1	Akshay	Finance	23
3	Balram	Sales	25
5	Manoj	Marketing	23
9	Naveen	Marketing	22
7	Parul	Finance	22
2	Ram	Marketing	24
6	Sheetal	Finance	24
10	Tarun	Finance	23

Ex No: 2	**Create a set of tables, add foreign key constraints and incorporate referential integrity**

Aim:

To create a set of tables, add foreign key constraints and incorporate referential integrity

Key Constraints in DBMS:

- Constraints or nothing but the rules that are to be followed while entering data into columns of the database table
- Constraints ensure that data entered by the user into columns must be within the criteria specified by the condition
- For example, if you want to maintain only unique IDs in the employee table or if you want to enter only age under 18 in the student table etc
- **We have 5 types of key constraints in DBMS**
    - **NOT NULL: ensures that the specified *column doesn't contain a NULL value.***
    - **UNIQUE: *provides a unique/distinct values* to specified columns.**
    - **DEFAULT: *provides a default value to a column* if none is specified.**
    - **CHECK: checks *for the predefined conditions before inserting* the data inside the table.**
    - **PRIMARY KEY: it *uniquely identifies a row* in a table.**
    - **FOREIGN KEY: ensures *referential integrity* of the relationship**

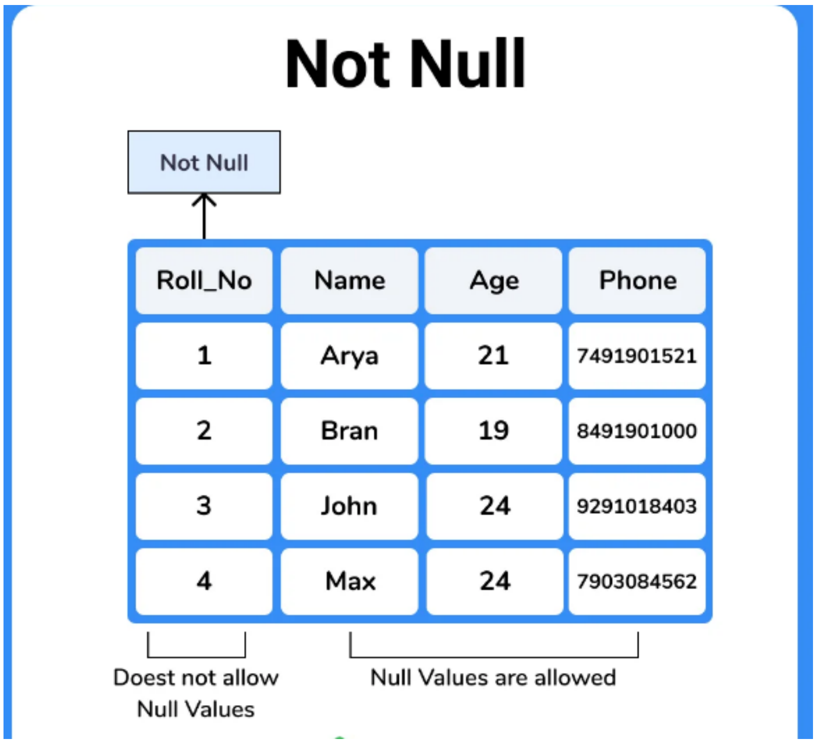
<u>Not Null</u>

- Null represents a record where data may be missing data or data for that record may be optional
- Once not null is applied to a particular column, you cannot enter null values to that column and restricted to maintain only some proper value other than null
- A not-null constraint cannot be applied at table level

<u>Example</u>

```
CREATE TABLE Orders (

   OrderID int NOT NULL,

   OrderNumber int NOT NULL,

   PersonID int,

   PRIMARY KEY (OrderID),

   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)

);
```

- In the above example, we have applied not null on three columns ID, name and age which means *whenever a record is entered using insert statement all three columns should contain a value other than null*
- We have two other columns address and salary, *where not null is not applied* which means that *you can leave the row as empty or use null value while inserting the record into the table*



Roll_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084562

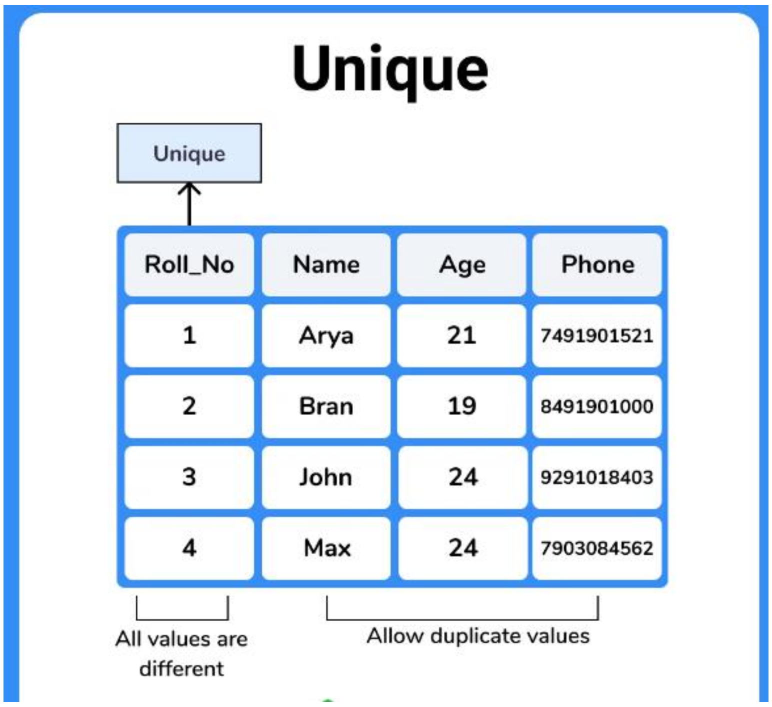
Unique

- **Sometimes we need to maintain only unique data in the column of a database table, this is possible by using a unique constraint**
- **Unique constraint ensures that all values in a column are unique**

Example

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

In the above example, ***as we have used unique constraint on ID column we are not supposed to enter the data that is already present,*** **simply no two ID values are same**



Roll_No	Name	Age	Phone
1	Arya	21	7491901521
2	Bran	19	8491901000
3	John	24	9291018403
4	Max	24	7903084562

DEFAULT

- Default clause in SQL is used to add default data to the columns
- When a column is specified as default with some value then all the rows will use the same value i.e each and every time while entering the data we need not enter that value
- But *default column value can be customized* i.e it can be overridden when inserting a data for that row based on the requirement.

Example for DEFAULT clause

The following SQL sets a DEFAULT value for the "city" column when the "emp" table is created:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE emp (

   ID int NOT NULL,

   LastName varchar(255) NOT NULL,

   FirstName varchar(255),

   Age int,

   City varchar(255) DEFAULT 'hyderabad'

);
```

- As a result, whenever you insert a new row each time you need not enter a value for this default column that is ***entering a column value for a default column is optional and if you don't enter the same value is considered that is used in the default clause***

Check

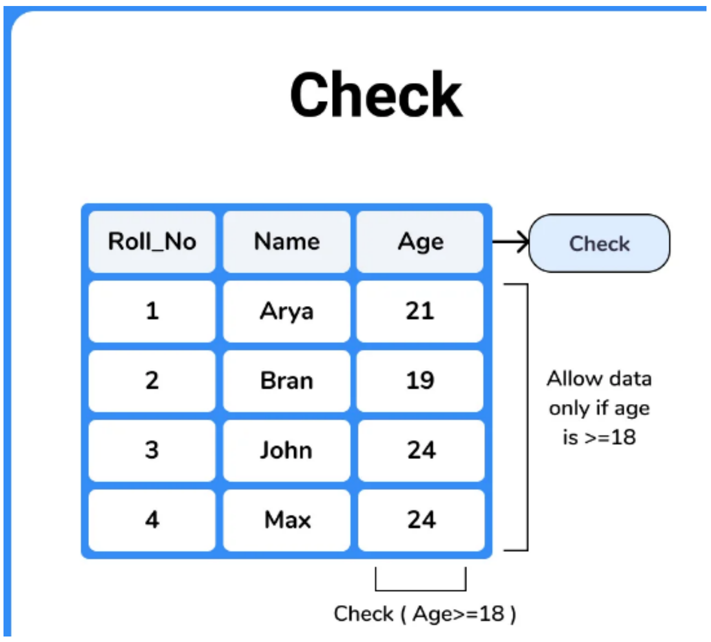
Suppose in real-time if you want to give access to an application only if the age entered by the user is greater than 18 this is done at the back-end by using a check constraint

Check constraint ensures that the data entered by the user for that column is within the range of values or possible values specified.

<u>Example for check constraint</u>

```
CREATE TABLE STUDENT (

   ID int ,

   Name varchar(255) ,

   Age int,

   CHECK (Age>=18)

);
```

- As we have used a ***check constraint as (Age>=18)*** which means ***values entered by the user for this age column while inserting the data must be less than or equal to 18*** otherwise an error is shown
- Simply, the only possible values that the ***age column will accept is [0 -17]***



Roll_No	Name	Age
1	Arya	21
2	Bran	19
3	John	24
4	Max	24

Primary Key

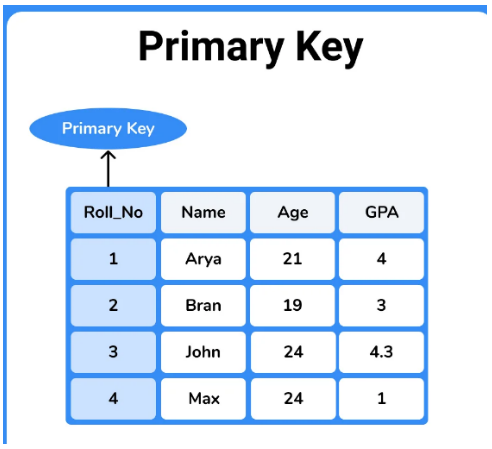
A primary key is a constraint in a table that uniquely identifies each row record in a database table by enabling one or more the columns in the table as the primary key.

Creating a primary key

A particular column is made as a primary key column by using the primary key keyword followed with the column name

```
CREATE TABLE EMP (
  ID   INT
  NAME VARCHAR (20)
  AGE  INT
  COURSE VARCHAR(10)
  PRIMARY KEY (ID)
);
```

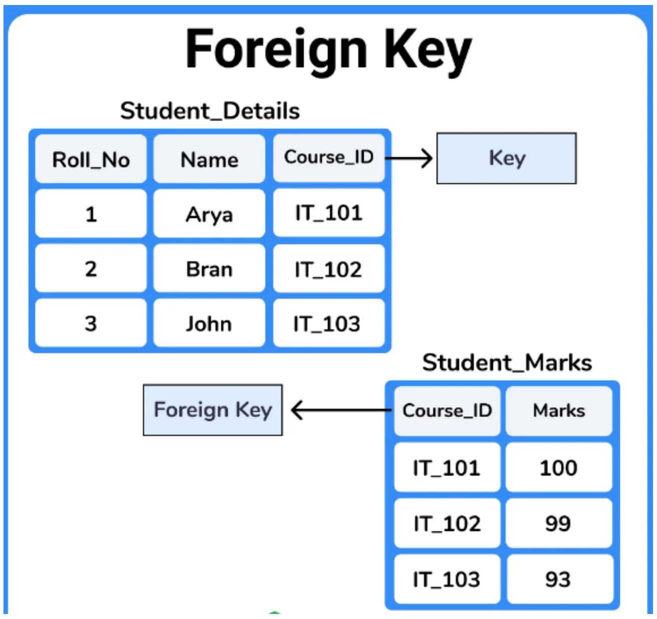
- Here we have used the primary key on ID column then ID column must contain unique values i.e ***one ID cannot be used for another student***.
- If you try to ***enter duplicate value while inserting in the row you are displayed with an error***
- Hence ***primary key will restrict you to maintain unique values and not null values in that particular column***



Roll_No	Name	Age	GPA
1	Arya	21	4
2	Bran	19	3
3	John	24	4.3
4	Max	24	1

Foreign Key

- The foreign key a constraint is a column or list of columns that points to the primary key column of another table
- The main purpose of the foreign key is only those values are allowed in the present table that will match the primary key column of another table.
- (*Sometimes you need to maintain and restrict only same data in a table that is exactly the same column data of another table, this purpose is served by using a foreign key*.)



Roll_No	Name	Course_ID
1	Arya	IT_101
2	Bran	IT_102
3	John	IT_103

Course_ID	Marks
IT_101	100
IT_102	99
IT_103	93

Example to create a foreign key

Reference Table

```
CREATE TABLE CUSTOMERS1(
  ID   INT ,
 DEPT VARCHAR (20)
  PRIMARY KEY (ID)
);
```

<u>Child Table</u>

```
CREATE TABLE CUSTOMERS2(
  ID   INT ,
  ADDRES VARCHAR (20)
  REFERENCES CUSTOMERS1(ID)
);
```

<u>CUSTOMERS1 table:</u>

ID	DEPT
65	Dairy
66	Snacks
67	Snacks

<u>CUSTOMERS2 table:</u>

ID	ADDRESS
65	Hyderabad
66	Chennai
67	Hyderabad

- ID column in the customers1 table is used as a foreign key in the customers2 table which means ***all the ID values in customers2 must exist in the customers1 table***
- An ID value that is ***not present in customers1 table is not allowed to be entered in the customers2 table ID column***
- ID column of the customers1 table contains values as 65 66 67 now ID column in customers2 table must contain only these values that is 65 66 67 ,***if the user enters other than this values in the ID column of the customers2 table it will raise an error because customers1 table id column is a foreign key in the customers2 table***
- Hence we can observe that a link is maintained between two tables that is if you want to enter any data in ***the foreign key column table then we must add the data in the primary key column of the parent table if it is not present***

Note:

- The column or list of the ***column that is used as foreign key*** in the present table ***must be a primary key in another table***
- The ***structure and data type*** of a PRIMARY KEY column of one table which used as a FOREIGN KEY in another table ***must be the same***
- The table containing the foreign key is called the ***child table***, and the table containing the candidate key is called the ***referenced or parent tabl***

Why foreign key?

- A foreign key is used to ***prevent activities that would destroy the link between tables***
- A foreign key ***prevents invalid data being inserted*** into the foreign key column because it restricts the user to enter only those values that are present in the primary key of another table.

Ex No: 3	Query the database tables using different 'where' clause conditions and also implement Aggregate functions

Aim:

To Query the database tables using different 'where' clause conditions and also implement Aggregate functions.

Aggregate functions in DBMS

- SQL provides a number of built-in functions to perform operations on data these functions are very much useful for performing mathematical calculations on table data
- Aggregate functions ***return a single value after performing calculations on a set of values***, here will discuss the five frequently used aggregate functions provided by SQL
- These aggregate functions are used with the SELECT statement at a time only one column can be applied with the function

General syntax

SELECT *functionname*(column_name)
FROM table_name
consider sample table emp

avg()

- This function ***returns the Arithmetic mean of all the values present in that column***

eid	name	age	salary
65	Trish	22	9000
66	Rishi	29	8000
67	Mahi	34	6000
68	Mani	44	10000
69	Puppy	35	8000

<u>SQL query to find average salary</u>

SELECT *avg*(salary) from Emp;

avg(salary)
8200

<u>count()</u>

- It ***returns the number of rows present in the table which can be either based upon a condition or without a condition***

<u>SQL query to count employees, satisfying specified condition is,</u>

SELECT *COUNT*(name) FROM Emp WHERE salary = 8000;

O/P

count(name)
2

<u>Example of COUNT (distinct)</u>

Consider the following **Emp** table

eid	**name**	**age**	**salary**
65	Trish	22	9000
66	Rishi	29	8000
67	Mahi	34	6000
68	Mani	44	10000
69	Puppy	35	8000

SQL query is,
SELECT *COUNT*(DISTINCT salary) FROM emp;

O/P

count(distinct salary)
4

max()

- It ***returns*** the maximum i.e the ***largest value among all the values present in that column***

SQL query to find the Maximum salary

SELECT *MAX*(salary) FROM emp;

O/P

MAX(salary)
10000

min()

- It ***returns*** the minimum value i.e the ***smallest numerical value among all the values present in that particular column***

SQL query to find minimum salary

SELECT ***MIN***(salary) FROM emp;

Result will be,

MIN(salary)
6000

sum()

- The sum function returns the arithmetic sum of all the values present in that column

<u>SQL query to find sum of salaries</u>

SELECT *SUM*(salary) FROM emp;

Result of above query is,

SUM(salary)
41000

Find the 2nd highest salary of an employee (interview question)

- This is the ***most commonly asked interview question***. Hence I recommend you remember this example
- This solution ***uses a sub query to first exclude the maximum salary*** from the data set and ***then again finds the maximum salary, which is effectively the second maximum salary from the Employee table.***

Select *max*(salary) from employee where salary < (select *max*(salary) from emp);

0/P

MAX(salary)
9000

<u>How it works</u>

- Here first we are ***creating a result set excluding first highest salary*** i.e max(salary)
- Again ***applying max(sal) on this result*** would Fetch the second highest salary obviously

Ex No: 4	Query the database tables and explore sub queries and simple join operations.

Aim:

To Query the database tables and explore sub queries and simple join operations.

Description:

- An **SQL Join statement** is used to combine data or rows from two or more tables based on a common field between them.
- A **sub query** is a query that is nested inside a SELECT, INSERT, UPDATE, or DELETE statement, or inside another sub query.
- **Joins and sub queries** are both used to combine data from different tables into a single result.

What is a Subquery?

A ***subquery*** is a nested query (inner query) that's used to filter the results of the outer query. Subqueries can be used as an alternative to joins. A subquery is typically nested inside the WHERE clause.

SQL Sub Query

A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rule:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

1. Subqueries with the Select Statement

SQL Subqueries are most frequently used with the Select statement.

Syntax

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
( SELECT column_name  from table_name WHERE ... );
```

Example

Consider the EMPLOYEE table have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

The subquery with a SELECT statement will be:

```
SELECT *
   FROM EMPLOYEE
   WHERE ID IN (SELECT ID
   FROM EMPLOYEE
   WHERE SALARY > 4500);
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
7	Jackson	25	Mizoram	10000.00

2. Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

```
INSERT INTO table_name (column1, column2, column3....)
SELECT *
FROM table_name
WHERE VALUE OPERATOR
```

Example

- Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.
- Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
INSERT INTO EMPLOYEE_BKP
  SELECT * FROM EMPLOYEE
  WHERE ID IN (SELECT ID
  FROM EMPLOYEE);
```

3. Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

```
UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
  (SELECT COLUMN_NAME
  FROM TABLE_NAME
  WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
UPDATE EMPLOYEE
  SET SALARY = SALARY * 0.25
  WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
    WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	1625.00

5	Kathrin	34	Bangalore	2125.00
6	Harry	42	China	1125.00
7	Jackson	25	Mizoram	10000.00

<u>4. Subqueries with the DELETE Statement</u>

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

<u>Syntax</u>

```
DELETE FROM TABLE_NAME
WHERE VALUE OPERATOR
  (SELECT COLUMN_NAME
  FROM TABLE_NAME
  WHERE condition);
```

<u>Example</u>

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

```
DELETE FROM EMPLOYEE
  WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
    WHERE AGE >= 29 );
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
7	Jackson	25	Mizoram	10000.00

Ex No: 5	Query the database tables and explore natural, equi and outer joins.

Aim:

To Query the database tables and explore natural, equi and outer joins.

Different Types of SQL JOINs

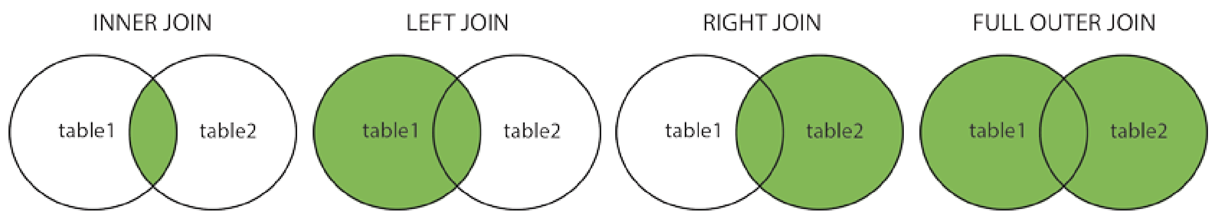
Here are the different types of the JOINs in SQL:

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



Sample Table

EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

PROJECT

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

1. INNER JOIN

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,

FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
INNER JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

2. LEFT JOIN

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
LEFT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	**DEPARTMENT**
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

3. RIGHT JOIN

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT

FROM EMPLOYEE

RIGHT JOIN PROJECT

ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	**DEPARTMENT**
Angelina	Testing
Robert	Development
Christian	Designing

Kristen	Development

4. FULL JOIN

Syntax

```
SELECT table1.column1, table1.column2, table2.column1,

FROM table1

FULL JOIN table2

ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT

FROM EMPLOYEE

FULL JOIN PROJECT

ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

Ex No: 6	Write user defined functions and stored procedures in SQL

Aim:

Description:

User Defined Functions in SQL

The UDF or User Defined Functions in SQL Server are like methods in any other programming language that accepts the parameters, performs complex calculations, and returns the result value.

Types of Functions in SQL Server

There are two types of SQL Server functions:

Built-in Functions

All the built-ins supported by Microsoft are called System functions. We don’t have to bother about the logic inside them because they cannot be modified. For example, Mathematical, Ranking, and String are some of the many built-in functions.

The aggregate functions to find the sum, minimum value, and an average value is mostly used in system methods. And finding the current system date and time is also the most frequent one.

User Defined Functions

SQL Server allows us to create our methods called user defined functions. For example, if we want to perform some complex calculations, then we can place them in a separate method and store it in the database. Whenever we need the calculation, we can call it. There are three types of SQL functions:

Scalar: It returns a single value. Generally, we have to define the body between BEGIN … END block, but for inline scalar function, you can omit them. We can use any data type as the return type except text, image, ntext, cursor, and timestamp.

Table Valued: It is a user defined function that returns a table.

Inline Table valued: It returns a table data type based on a single SELECT Statement.

Advantages of UDFs

1. The SQL Server User defined functions prevent us from writing the same logic multiple times.
2. Within the Database, you can create the method once and call it n number of times.
3. They reduce the compilation time of queries by catching the execution plan and reusing them.
4. This UDF can help us to separate the complex calculations from the regular query so that we can understand and debug the query quicker and better.
5. It reduces the network traffic because of its cache plan
6. They are also used in the WHERE Clause as well. By this, we can limit the number of rows sent to the client.

SQL User Defined Functions Syntax

The syntax of the SQL Server functions or UDF is

```
CREATE FUNCTION Name(@Parameter_Name Data_type,
                      .... @Parameter_Name Data_type
                    )
RETURNS Data_Type
AS
  BEGIN
```

```
    -- Function_Body


    RETURN Data
  END
```

- Return_Type:
    1. Data Type: Please specify the data type of return value. For example, VARCHAR, INT, FLOAT, etc.
    2. Data: Please specify the return value, and it should match the Data Type. It can be a single value or Table.
- Name: You can specify any name you wish to give other than the system reserved keywords. Please try to use meaningful names so that you can identify them easily.
- @Parameter_Name: Every method accepts zero or more parameters; it completely depends upon the user requirements. While declaring the parameters, don't forget the appropriate data type. For example (@name VARCHAR(50), @number INT)
- Function_Body: Any query or any complex mathematical calculations you want to implement in this particular method.

Optimal Solution (Alternative Solution for above question)

The concept of functions in SQL is similar to other programming languages like Python. The major difference being the way they are implemented. There are two main types of user-defined functions in SQL based on the data they return:

1. **Scalar functions:** These types of functions return a single value, i.e. float, int, varchar, date time, etc.
2. **Table-Valued functions:** These functions return tables.

Creating functions

Scalar functions

Below is the definition of a simple function. It takes in two numbers and returns their sum.

Since this function returns a number, it is a scalar function.

```
CREATE FUNCTION scalar_func
    (
          @a AS INT, -- parameter a
          @b AS INT -- parameter b
    )
    RETURNS INT -- return type
    AS
    BEGIN
          RETURN @a + @b -- return statement
    END;
```

- We use the Create function command to define functions. It is followed by the name of the function. In the above example, the name of the function is scalar_func.
- We need to declare the parameters of the function in the following format.

@VariableName AS Data Type

In our above example, we have defined two integer parameters a and b.

- The return type of the result has to be mentioned below the definition of the parameters. In the above example, we are returning the sum that is an integer.
- After the return statements, we create a BEGIN ... END block that contains the logic of our function. Although in this case, we have a single return statement, we don’t need a BEGIN ... END block.

Table-valued functions

Before creating a table-valued function, we will create a simple table.

```
-- Creating new table
    CREATE TABLE TEST(
          num1 INT,
          num2 INT
    );

    -- Inserting values into new table
    INSERT INTO TEST
    VALUES
    (1,2),
    (2,3),
      (4,5);
```

The table contains 2 columns. We will create a function that returns a new table with an extra column. This extra column will contain the sum of numbers in the column num1 and column num2.

```
CREATE FUNCTION table_valued_func()
    RETURNS TABLE
    AS
    RETURN
          -- statement to calculate sum
          SELECT num1 , num2, num1 + num2 AS 'SUM'
          FROM TEST;
```

- The function above does not take in any parameter.
- The SQL statement simply calculates the sum and stores it in a new column named SUM.