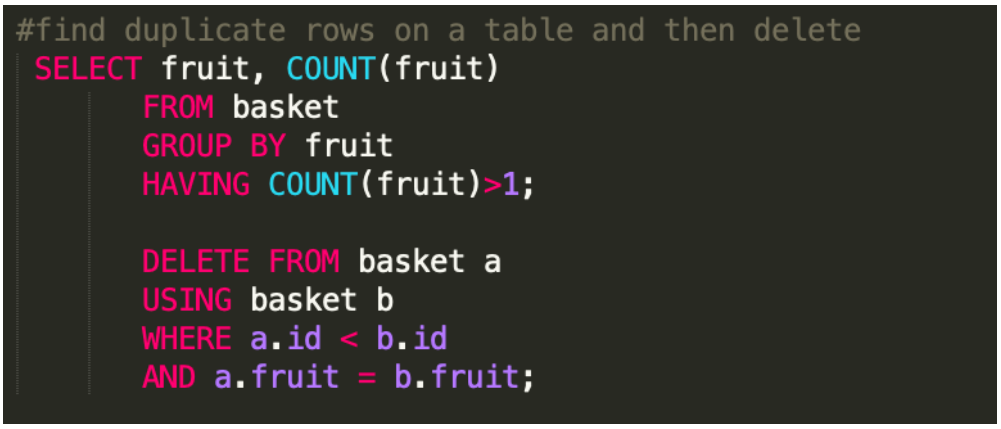
1. What is the difference between the HAVING and WHERE clause in SQL?

The WHERE clause allows you to specify a certain condition on columns of data to return rows that match the condition criteria. I.e. WHERE amount > 100
The HAVING clause allows you to do additional filtering after an aggregation function. Usually after the GROUP BY statement in SQL. i.e. HAVING SUM(sales) >= 1200

2. What is the difference between GROUP BY and ORDER BY in SQL?

GROUP BY function allows you to aggregate data and apply function to better display how that data is distributed per category. i.e. GROUP BY reporting year. ORDER BY allows you to do additional order filtering on the data in either ASC or DESC order. Usually the final statement after all others.

3. Write a query to DELETE duplicate rows from a table ?



```
#find duplicate rows on a table and then delete
 SELECT fruit, COUNT(fruit)
        FROM basket
        GROUP BY fruit
        HAVING COUNT(fruit)>1;

        DELETE FROM basket a
        USING basket b
        WHERE a.id < b.id
        AND a.fruit = b.fruit;
```

4. What are the SQL JOINS?

INNER JOIN: Retrives records that have matching values in both tables in the join. Probably most widely used.

SELECT name, address
FROM Customer
INNER JOIN address
ON customer.address_id = address.address_id;

FULL OUTHER JOIN: Retrieves all matched and unmatched rows from tables on both sides of the join.
SELECT *
FROM Customer
FULL OUTER JOIN address
ON customer.address_id = address.address_id;

LEFT OUTER JOIN: Retrieves records that are in the left table , if there are no matches on the right tables the results are shown as null on the right side of the join.

SELECT * FROM customer
LEFT OUTER JOIN address
ON customer.customer_id = address.address_id;

RIGHT OUTER JOIN: Opposite of the above.

CROSS JOIN: Used if we want to combine all rows of one table with all rows of another table. If want to know all possible combinations between two tables that have related information.

5. What are constraints in SQL?

Constraints: are used to specify the rules concerning data in the table. Used to prevent invalid data from being entered into a database, ensuring accuracy and reliability.
It can be applied for a single or multiple fields in sql table during creation of table or after creating or using the ALTER TABLE command.

- **NOT NULL Constraint:** column cannot have a NULL.
- **UNIQUE Constraint:** ensure values are unique in the table.
- **PK Constraint:** unquely identifies each row in a table.
- **FK Constraint:** constrains data based on columns in other tables.
- **Check Constraint:** verifies that all values in a field satisfies a condition.
- **Default Constraint:** Automatically assigns a default values if no value has been specified.

PK constraint Example:

```
ALTER TABLE CLIENT_MONTHLY_SF
ADD CONSTRAINT CLIENT_MONTHLY_SF_PK
PRIMARY KEY (CLIENT_KY, PAYER_KY);
```

6. What are the keys in RDBMS?

Primary Key: Uniquely identifies each row in a table. It must contain unique values and has an implicit NOT NULL constraint.
Foreign Key: A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key. A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.
Composite Key: A composite key is a combination of two or more columns in a table that can be used to uniquely identify each row in the table when the columns are combined uniqueness is guaranteed, but when it taken individually it does not guarantee uniqueness. ... Columns that make up the composite key can be of different data types.
Surrogate Key: An attribute or set of attribute that be declared as the primary key of entity. A surrogate is used where there is no business or natural key.
Natural Key: A naturally occurring descriptor of an entity and one of the entities.
Candidate Key: CANDIDATE KEY is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key.

7. What is Normalization?

Normalization: is the process of refining tables, keys, columns and relationships to create an efficient database. This is a design process that reduces data redundancy of data from the database . Simply divides larger tables into smaller tables and link them using logical relationships.

There are two reasons for the normalization process:

- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.
- Goals is reduce the amount of space a database consumes and ensure that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into **normal forms;**

Think of form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure so that it complies with the rules of first normal form, then second normal form, and finally third normal form.

It's your choice to take it further and go to fourth normal form, fifth normal form, and so on, but generally speaking, **third normal form is enough.**

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

First normal form (1NF)
sets the very basic rules for an organized database:
• Define the data items required, because they become the columns in a table. Place related data items in a table.
• Ensure that there are no repeating groups of data.
• Ensure that there is a primary key.

1. First Rule of 1NF: You must define the data items. This means looking at the data to be stored, organizing the data into columns, defining what type of data each column contains,
and finally putting related columns into their own table. For example, you put all the columns relating to locations of meetings in the Location table, those relating to members in the MemberDetails table, and so on.
2. Second Rule of 1NF: The next step is ensuring that there are no repeating groups of data.
3. Third Rule of 1NF: The final rule of the first normal form, create a primary key for each table which we have already created.

Second Normal Form (2NF)
Second normal form states that it should meet all the rules for 1NF and there must be no partial dependences of any of the columns on the primary key

Third Normal Form (3NF)
A table is in third normal form when the following conditions are met:
• It is in second normal form.
• All nonprimary fields are dependent on the primary key. The dependency of nonprimary fields is between the data. For example, in the below table, street name, city, and state are unbreakably bound to the zip code.

What is the difference between DELETE, TRUNCATE and DROP

- DELETE statement removes rows from the table. DELETE Query is used to delete the existing records from a table. You can use the WHERE clause with a DELETE query to delete.
- DROP table command is used to delete a table and all rows in the table. ... Deleting all of the records in the table leaves the table including column and constraint information. Dropping the table removes the table definition as well as all of its rows.
- TRUNCATE TABLE removes all rows from a table, but the table structure and its columns, constraints, indexes, and so on remain.

Level 2

8. Difference between CHAR and Varchar?

- VARCHAR is variable length string data type, while CHAR is fixed length.
- CHAR is a fixed length string data type, so any remaining space in the field is padded with blanks.
- CHAR takes up 1 byte per character. Often used for specifying indicator Y/N VARCHAR is a variable length string data type, so it holds only the characters you assign to it.

9. Difference between JOIN and UNION?

UNION in SQL is used to combine the result-set of two or more SELECT statements.
The data combined using UNION statement puts results into new distinct rows. JOIN combines data from many tables based on a matched condition between them.

10. Difference between UNION ALL and UNION?

UNION retrieves only distinct records from all queries or tables, whereas UNION ALL returns all the records retrieved by queries.
Performance of UNION ALL is higher than UNION.

11. Difference between IN & EXISTS?

EXISTS is used to determine if any values are returned or not. Whereas, IN can be used as a multiple OR operator. If the sub-query result is large, then EXISTS is faster than IN. Once the single positive condition is met in the EXISTS condition then the SQL Engine will stop the process.

12. How to create empty table with same structure as another table?

```
SELECT * INTO students_copy
FROM students
WHERE 1 = 2;
```

13. What is pattern matching in SQL?

Pattern matching uses wildcard% searching with LIKE operator to find a match in a string.
% matches sequence of chars, underscore __ matches single characters.

14. What is MINUS and INTERSECT in SQL?

INTERSECT: When user wants to fetch the common records from the two different tables then INTERSECT operator come in to picture. Intersect operator fetches the record which are common between 2 tables.

MINUS: When user wants to fetch the record from one table only and not the common records between two tables user needs to use MINUS operator. MINUS operator selects all the rows from first table but not from second table. It eliminates duplicate rows from first and second table. It removes the results from second table and always considered first table only.

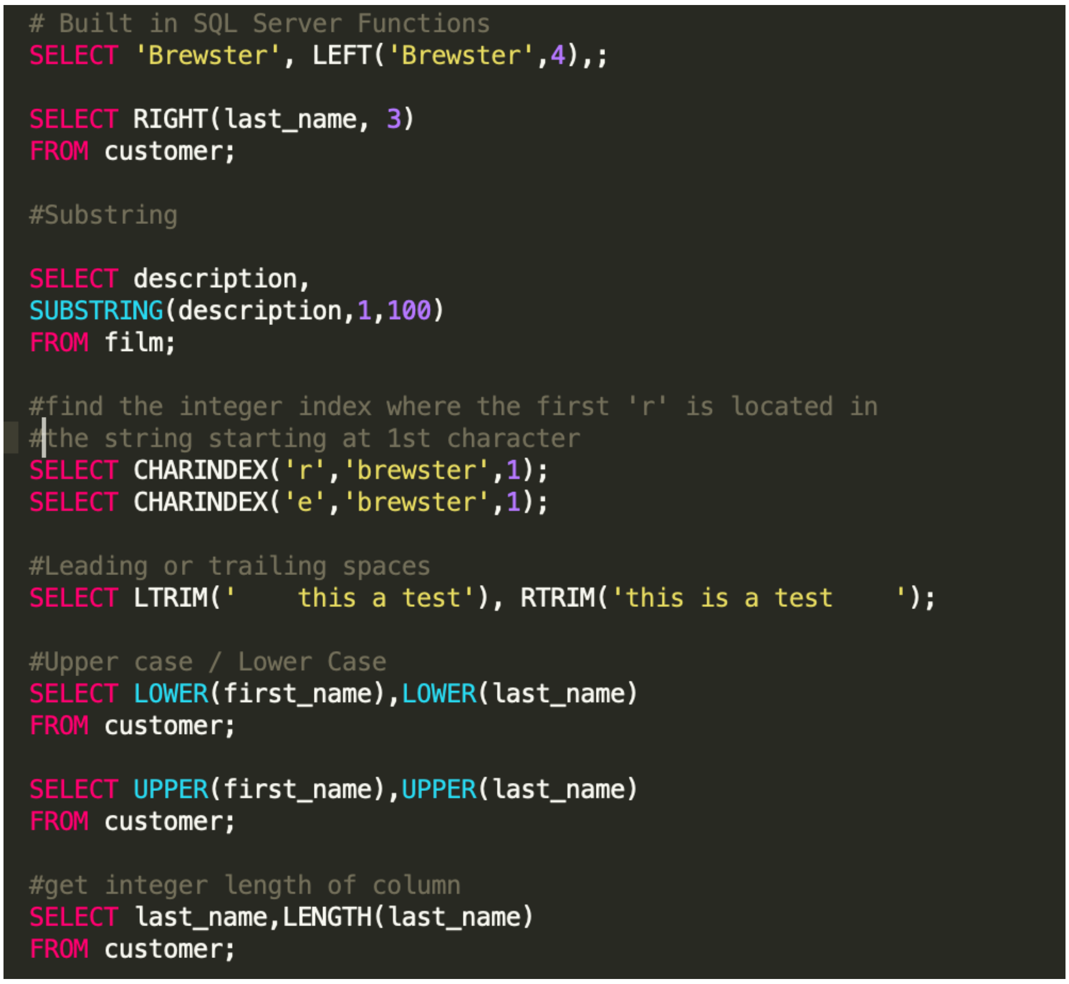
15. What is a view?

A view is like a virtual table based on a result-set of an SQL statement The fields in a view are fields from one or more real tables in the database.

- Views can represent a subset of the data contained in a table.
- Views can join and simplify multiple tables into a single virtual table.

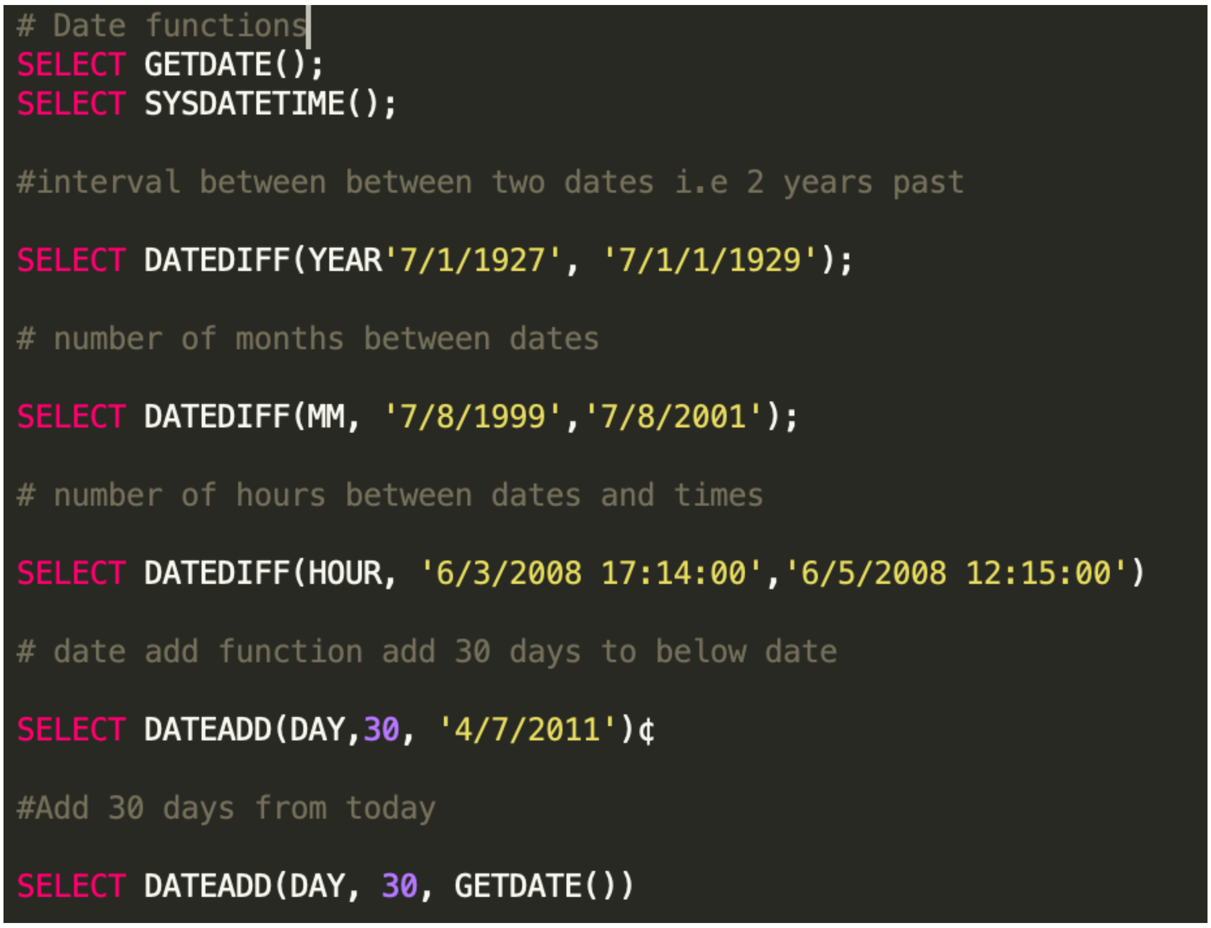
```
CREATE VIEW vTopSalesByQty
AS
    SELECT TOP 3 product_id, product_name, SUM(sales)
    FROM sales;
```

16. What are the Character manipulation functions?



```
# Built in SQL Server Functions
SELECT 'Brewster', LEFT('Brewster',4),;

SELECT RIGHT(last_name, 3)
FROM customer;

#Substring

SELECT description,
SUBSTRING(description,1,100)
FROM film;

#find the integer index where the first 'r' is located in
#the string starting at 1st character
SELECT CHARINDEX('r','brewster',1);
SELECT CHARINDEX('e','brewster',1);

#Leading or trailing spaces
SELECT LTRIM('    this a test'), RTRIM('this is a test    ');

#Upper case / Lower Case
SELECT LOWER(first_name),LOWER(last_name)
FROM customer;

SELECT UPPER(first_name),UPPER(last_name)
FROM customer;

#get integer length of column
SELECT last_name,LENGTH(last_name)
FROM customer;
```

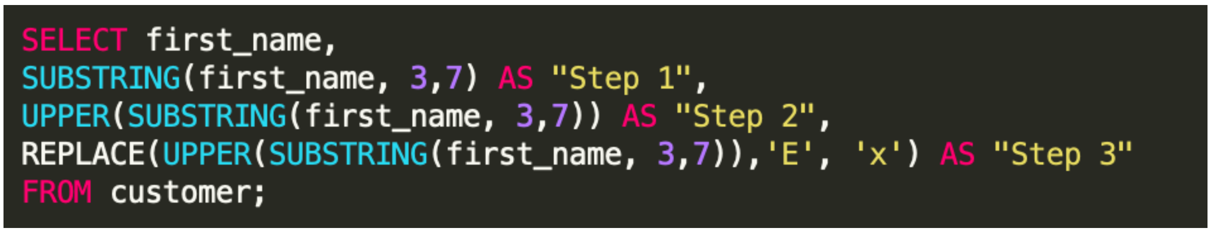
17. **What Are the Date Manipulation Functions?**



```
# Date functions
SELECT GETDATE();
SELECT SYSDATETIME();

#interval between between two dates i.e 2 years past

SELECT DATEDIFF(YEAR'7/1/1927', '7/1/1/1929');

# number of months between dates

SELECT DATEDIFF(MM, '7/8/1999','7/8/2001');

# number of hours between dates and times

SELECT DATEDIFF(HOUR, '6/3/2008 17:14:00','6/5/2008 12:15:00')

# date add function add 30 days to below date

SELECT DATEADD(DAY,30, '4/7/2011')¢

#Add 30 days from today

SELECT DATEADD(DAY, 30, GETDATE())
```

18. **What are Nested Functions?**

SQL query that nests function inside other functions to obtain a desired result.



```
SELECT first_name,
SUBSTRING(first_name, 3,7) AS "Step 1",
UPPER(SUBSTRING(first_name, 3,7)) AS "Step 2",
REPLACE(UPPER(SUBSTRING(first_name, 3,7)),'E', 'x') AS "Step 3"
FROM customer;
```

19. What is a scaler function?

SQL Server scalar function takes one or more parameters and returns a single value. The scalar functions help you simplify your code. For example, you may have a complex calculation that appears in many queries.

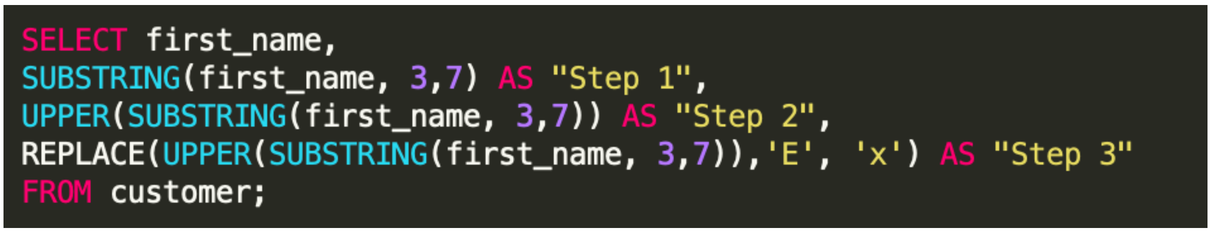
If a column function's argument is a scalar function, the scalar function must include a reference to a column. For example, if you want to know the last year that any project begins and the last year that any project completes,

Nesting scalar functions within scalar functions

```
SELECT SUBSTR((CHAR(INTDATE, USA)),1,5)
  FROM Q.INTERVIEW
  WHERE MANAGER = 140
```

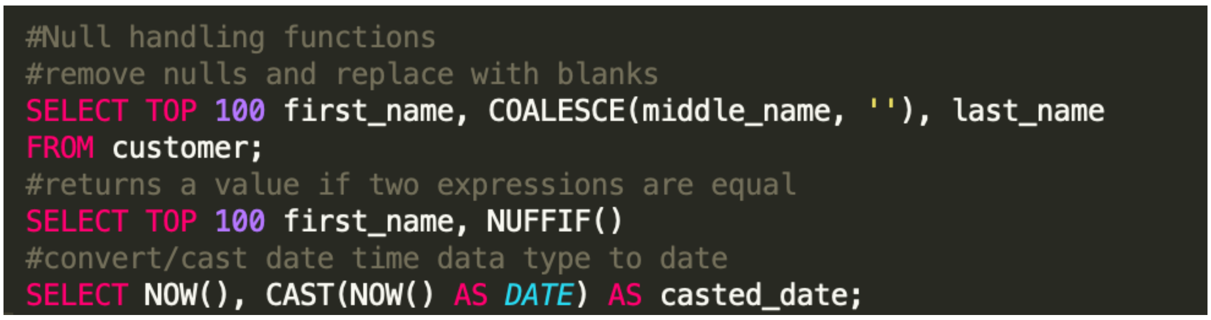
Nesting scalar functions within column functions

```
SELECT MAX(YEAR(STARTD)), MAX(YEAR(ENDD))
  FROM Q.PROJECT
```



```
SELECT first_name,
SUBSTRING(first_name, 3,7) AS "Step 1",
UPPER(SUBSTRING(first_name, 3,7)) AS "Step 2",
REPLACE(UPPER(SUBSTRING(first_name, 3,7)),'E', 'x') AS "Step 3"
FROM customer;
```

20. How to Handle NULLS with Null handling functions?



```
#Null handling functions
#remove nulls and replace with blanks
SELECT TOP 100 first_name, COALESCE(middle_name, ''), last_name
FROM customer;
#returns a value if two expressions are equal
SELECT TOP 100 first_name, NUFFIF()
#convert/cast date time data type to date
SELECT NOW(), CAST(NOW() AS DATE) AS casted_date;
```

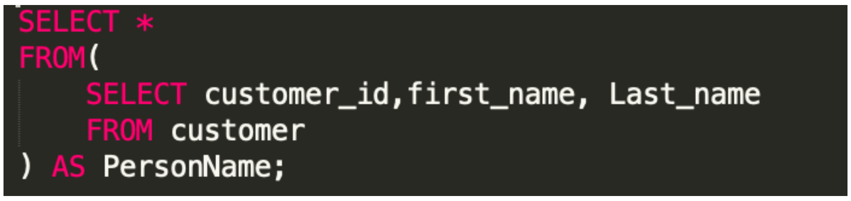
21. What are Derived Tables in SQL

A derived table is a technique for creating a temporary set of records which can be used within another query in SQL. You can use derived tables to shorten long queries, or even just to break a complex process into logical steps.

A derived table is a SQL query that defines a set of business logic, returns reduced amounts of data, and can include complex calculations and data transformations. You can use derived tables to improve the performance of analytic queries, encapsulate business logic, ease the burden of maintaining complex queries, and facilitate logic that can't be easily achieved in a single query.

One of the most common uses of derived tables is pre-aggregating data. Suppose, for example, you have a table that contains frequent sales transactions, and you want to see a report of sales summarized by user. If the data in your data warehouse is updated in near real time, it would make sense to calculate the appropriate aggregation using a CTE, because the results could change with each query execution, as new data is added.

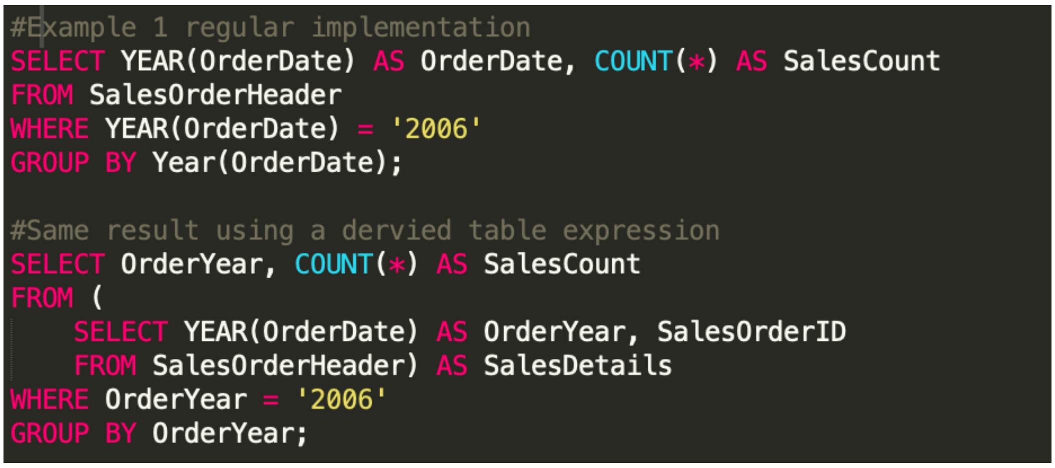
Ex 1:



```sql
SELECT *
FROM(
    SELECT customer_id,first_name, Last_name
    FROM customer
) AS PersonName;
```

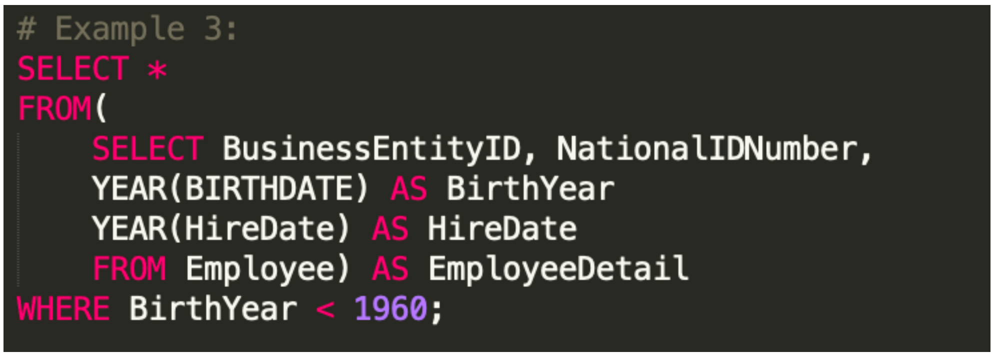
Below Derived table example to show the difference between a regular table query and a derived table implementation, derived tables improves performance as they not replicate the functions i.e. YEAR() for each clause in a query:

Ex 2:



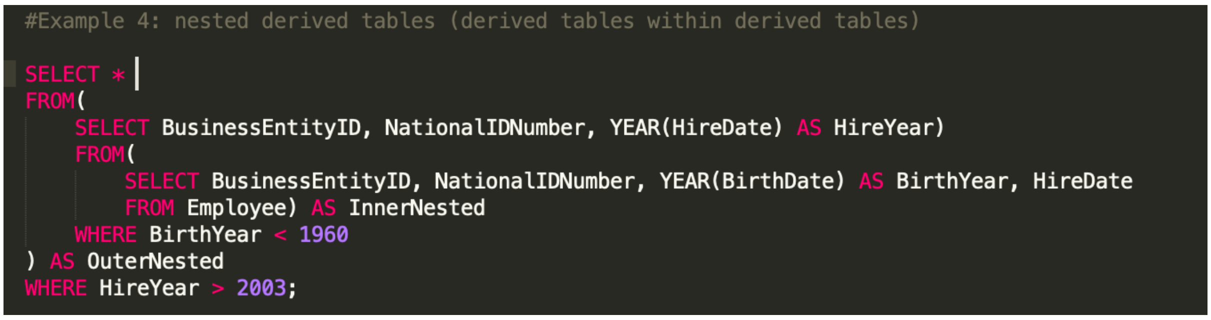
```sql
#Example 1 regular implementation
SELECT YEAR(OrderDate) AS OrderDate, COUNT(*) AS SalesCount
FROM SalesOrderHeader
WHERE YEAR(OrderDate) = '2006'
GROUP BY Year(OrderDate);

#Same result using a dervied table expression
SELECT OrderYear, COUNT(*) AS SalesCount
FROM (
    SELECT YEAR(OrderDate) AS OrderYear, SalesOrderID
    FROM SalesOrderHeader) AS SalesDetails
WHERE OrderYear = '2006'
GROUP BY OrderYear;
```

Ex 3:



```sql
# Example 3:
SELECT *
FROM(
    SELECT BusinessEntityID, NationalIDNumber,
    YEAR(BIRTHDATE) AS BirthYear
    YEAR(HireDate) AS HireDate
    FROM Employee) AS EmployeeDetail
WHERE BirthYear < 1960;
```

Ex 4: Nested Derived Tables



```
#Example 4: nested derived tables (derived tables within derived tables)

SELECT * 
FROM(
    SELECT BusinessEntityID, NationalIDNumber, YEAR(HireDate) AS HireYear)
    FROM(
        SELECT BusinessEntityID, NationalIDNumber, YEAR(BirthDate) AS BirthYear, HireDate
        FROM Employee) AS InnerNested
    WHERE BirthYear < 1960
) AS OuterNested
WHERE HireYear > 2003;
```

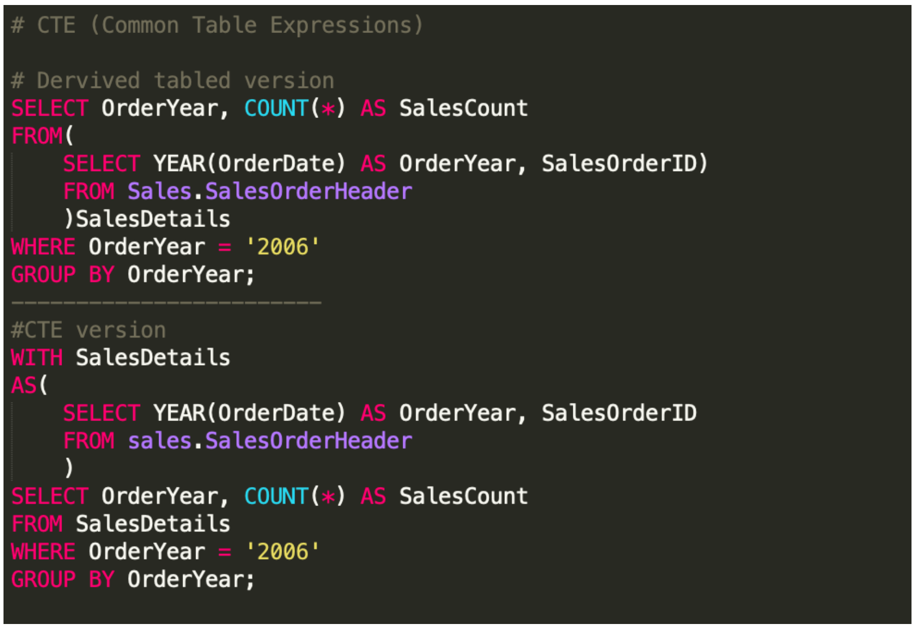
22. **What are CTE (Common Table Expressions) or WITH**

CTE stands for common table expression. A CTE allows you to define a temporary named result set thats available temporarily in the execution scope of a statement such as **SELECT, INSERT, UPDATE, DELETE, or MERGE.**

What are the benefits of using a CTE over derived tables?
Derived tables are subqueries that are used in the FROM clause instead of named **tables**. I like **using** CTEs **over derived tables** because CTEs are so much easier to read. **Derived tables** can be nested and often are several layers deep, becoming difficult to read and understand.

We prefer to use common table expressions rather than to use subqueries because common table expressions are more readable. We also use CTE in the queries that contain analytic functions (or window functions)

CTE Example 1:

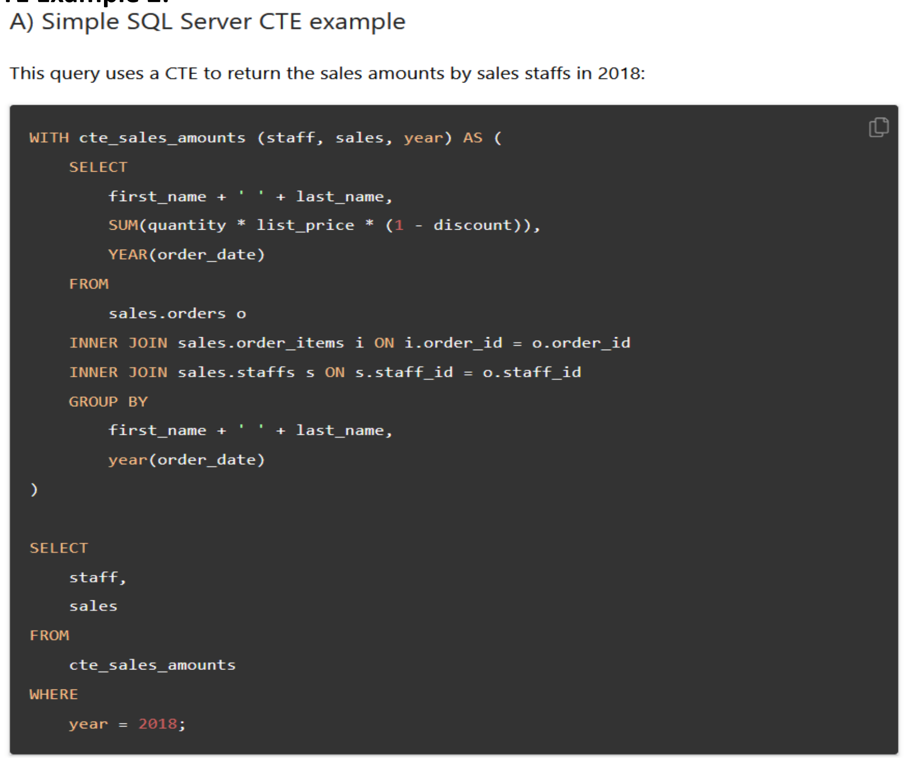


```
# CTE (Common Table Expressions)

# Dervived tabled version
SELECT OrderYear, COUNT(*) AS SalesCount
FROM(
    SELECT YEAR(OrderDate) AS OrderYear, SalesOrderID)
    FROM Sales.SalesOrderHeader
    )SalesDetails
WHERE OrderYear = '2006'
GROUP BY OrderYear;
------------------------
#CTE version
WITH SalesDetails
AS(
    SELECT YEAR(OrderDate) AS OrderYear, SalesOrderID
    FROM sales.SalesOrderHeader
    )
SELECT OrderYear, COUNT(*) AS SalesCount
FROM SalesDetails
WHERE OrderYear = '2006'
GROUP BY OrderYear;
```

In this syntax:

- First, specify the expression name (expression_name) to which you can refer later in a query.
- Next, specify a list of comma-separated columns after the expression_name. The number of columns must be the same as the number of columns defined in the CTE_definition.
- Then, use the AS keyword after the expression name or column list if the column list is specified.
- After, define a SELECT statement whose result set populates the common table expression.
- Finally, refer to the common table expression in a query (SQL_statement) such as SELECT, INSERT, UPDATE, DELETE, or MERGE.

CTE Example 2:

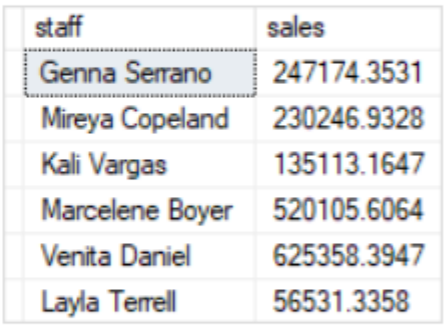


A) Simple SQL Server CTE example

This query uses a CTE to return the sales amounts by sales staffs in 2018:

```sql
WITH cte_sales_amounts (staff, sales, year) AS (
    SELECT
        first_name + ' ' + last_name,
        SUM(quantity * list_price * (1 - discount)),
        YEAR(order_date)
    FROM
        sales.orders o
    INNER JOIN sales.order_items i ON i.order_id = o.order_id
    INNER JOIN sales.staffs s ON s.staff_id = o.staff_id
    GROUP BY
        first_name + ' ' + last_name,
        year(order_date)
)

SELECT
    staff,
    sales
FROM
    cte_sales_amounts
WHERE
    year = 2018;
```

The following picture shows the result set:



staff	sales
Genna Serrano	247174.3531
Mireya Copeland	230246.9328
Kali Vargas	135113.1647
Marcelene Boyer	520105.6064
Venita Daniel	625358.3947
Layla Terrell	56531.3358

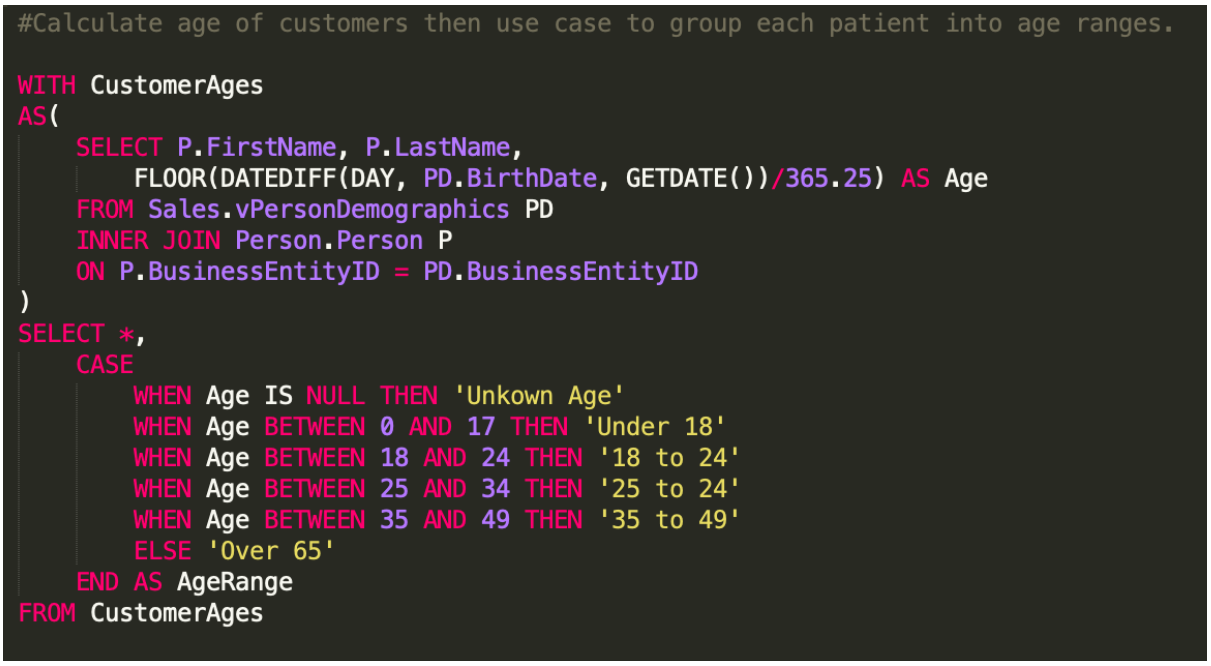
In this example:

- First, we defined cte_sales_amounts as the name of the common table expression. the CTE returns a result that that consists of three columns staff, year, and sales derived from the definition query.
- Second, we constructed a query that returns the total sales amount by sales staff and year by querying data from the orders, order_items and staffs tables.
- Third, we referred to the CTE in the outer query and select only the rows whose year are 2018.

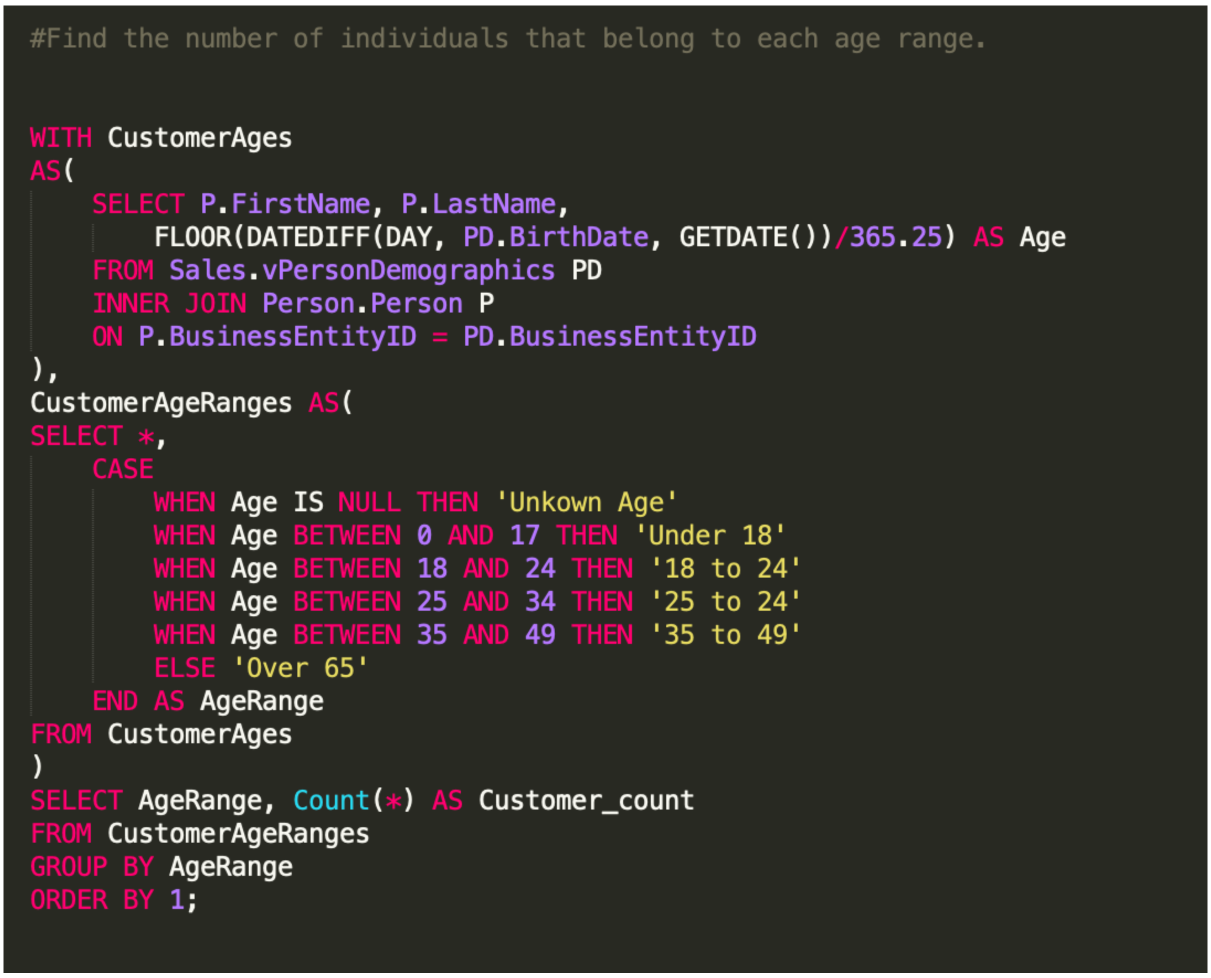
Noted that this example is solely for the demonstration purpose to help you gradually understand how common table expressions work. There is a more optimal way to achieve the result without using CTE.

23. What is CASE in SQL?

Calculate age of customers then use case to group each patient into age ranges:



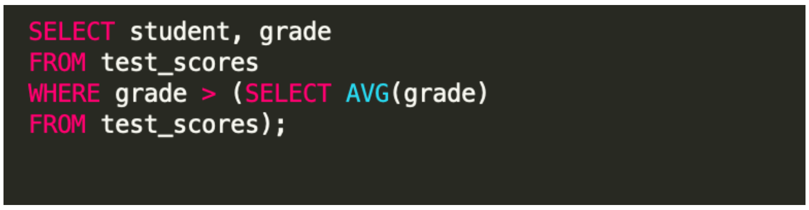
```sql
#Calculate age of customers then use case to group each patient into age ranges.

WITH CustomerAges
AS(
    SELECT P.FirstName, P.LastName,
        FLOOR(DATEDIFF(DAY, PD.BirthDate, GETDATE())/365.25) AS Age
    FROM Sales.vPersonDemographics PD
    INNER JOIN Person.Person P
    ON P.BusinessEntityID = PD.BusinessEntityID
)
SELECT *,
    CASE
        WHEN Age IS NULL THEN 'Unkown Age'
        WHEN Age BETWEEN 0 AND 17 THEN 'Under 18'
        WHEN Age BETWEEN 18 AND 24 THEN '18 to 24'
        WHEN Age BETWEEN 25 AND 34 THEN '25 to 24'
        WHEN Age BETWEEN 35 AND 49 THEN '35 to 49'
        ELSE 'Over 65'
    END AS AgeRange
FROM CustomerAges
```



```sql
#Find the number of individuals that belong to each age range.


WITH CustomerAges
AS(
    SELECT P.FirstName, P.LastName,
        FLOOR(DATEDIFF(DAY, PD.BirthDate, GETDATE())/365.25) AS Age
    FROM Sales.vPersonDemographics PD
    INNER JOIN Person.Person P
    ON P.BusinessEntityID = PD.BusinessEntityID
),
CustomerAgeRanges AS(
SELECT *,
    CASE
        WHEN Age IS NULL THEN 'Unkown Age'
        WHEN Age BETWEEN 0 AND 17 THEN 'Under 18'
        WHEN Age BETWEEN 18 AND 24 THEN '18 to 24'
        WHEN Age BETWEEN 25 AND 34 THEN '25 to 24'
        WHEN Age BETWEEN 35 AND 49 THEN '35 to 49'
        ELSE 'Over 65'
    END AS AgeRange
FROM CustomerAges
)
SELECT AgeRange, Count(*) AS Customer_count
FROM CustomerAgeRanges
GROUP BY AgeRange
ORDER BY 1;
```

24. What are Subquery + EXISTS

How can we can a list of students who scored better than the average grade?
Using a subquery can help us get the result in a single query request.



```
SELECT student, grade
FROM test_scores
WHERE grade > (SELECT AVG(grade)
FROM test_scores);
```

25. What are user-defined functions ?

LEVEL 3 - MS SQL SERVER

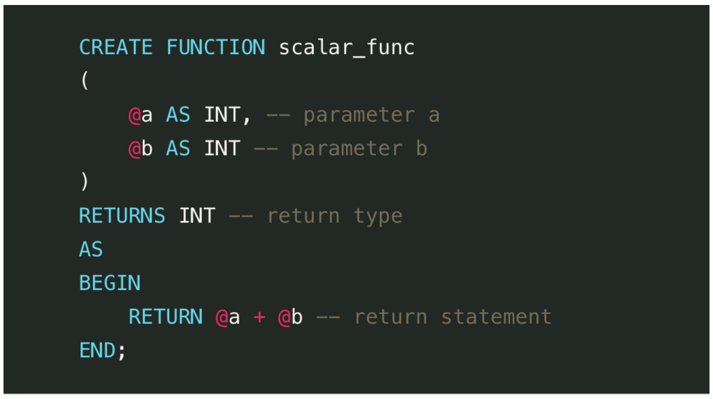
SQL Server **user-defined functions** are routines that accept parameters, perform an action, such as a complex calculation, and return the result of that action as a value. The return value can either be a single scalar value or a result set.

- They allow modular programming.
- They allow faster execution. Similar to stored procedures, Transact-SQL user-defined functions reduce the compilation cost of Transact-SQL code by caching the plans and reusing them for repeated executions.

Types Of User Defined Functions

Scalar Function
User-defined scalar functions return a single data value of the type defined in the RETURNS clause. For an inline scalar function, the returned scalar value is the result of a single statement. For a multi-statement scalar function, the function body can contain a series of Transact-SQL statements that return the single value. The return type can be any data type except text, ntext, image, cursor, and timestamp. Examples.
Table-Valued Functions
User-defined table-valued functions return a table data type. For an inline table-valued function, there is no function body; the table is the result set of a single SELECT statement. Examples.
System Functions
SQL Server provides many system functions that you can use to perform a variety of operations. They cannot be modified. For more information, see Built-in Functions (Transact-SQL), System Stored Functions (Transact-SQL), and Dynamic Management Views and Functions (Transact-SQL).

User-Defined Scaler Function example;
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.

Stored procedure in SQL Server

A stored procedure in SQL Server is a group of one or more Transact-SQL statements or a reference to a Microsoft .NET Framework common runtime language (CLR) method. Procedures resemble constructs in other programming languages because they can:

- Accept input parameters and return multiple values in the form of output parameters to the calling program.
- Contain programming statements that perform operations in the database. These include calling other procedures.
- Return a status value to a calling program to indicate success or failure (and the reason for failure).

Benefits of Using Stored Procedures

The following list describes some benefits of using procedures.

Reduced server/client network traffic
The commands in a procedure are executed as a single batch of code. This can significantly reduce network traffic between the server and client because only the call to execute the procedure is sent across the network. Without the code encapsulation provided by a procedure, every individual line of code would have to cross the network.

Stronger security
Multiple users and client programs can perform operations on underlying database objects through a procedure, even if the users and programs do not have direct permissions on those underlying objects. The procedure controls what processes and activities are performed and protects the underlying database objects. This eliminates the requirement to grant permissions at the individual object level and simplifies the security layers.

Types of Stored Procedures

User-defined
A user-defined procedure can be created in a user-defined database or in all system databases except the **Resource** database. The procedure can be developed in either Transact-SQL or as a reference to a Microsoft .NET Framework common runtime language (CLR) method.

Temporary
Temporary procedures are a form of user-defined procedures. The temporary procedures are like a permanent procedure, except temporary procedures are stored in **tempdb**. There are two types of temporary procedures: local and global. They differ from each other in their names, their visibility, and their availability. Local temporary procedures have a single number sign (#) as the first character of their names; they are visible only to the current user connection, and they are deleted when the connection is closed. Global temporary procedures have two number signs (##) as the first two characters of their names; they are visible to any user after they are created, and they are deleted at the end of the last session using the procedure.

DML triggers is a special type of stored procedure that automatically takes effect when a data manipulation language (DML) event takes place that affects the table or view defined in the trigger. DML events include INSERT, UPDATE, or DELETE statements. DML triggers can be used to enforce business rules and data integrity, query other tables, and include complex Transact-SQL statements. The trigger and the statement that fires it are treated as a single transaction, which can be rolled back from within the trigger. If a severe error is detected (for example, insufficient disk space), the entire transaction automatically rolls back.

DML Trigger Benefits

DML triggers are similar to constraints in that they can enforce entity integrity or domain integrity. In general, entity integrity should always be enforced at the lowest level by indexes that are part of PRIMARY KEY and UNIQUE constraints or are created independently of constraints. Domain integrity should be enforced through CHECK constraints, and referential integrity (RI) should be enforced through FOREIGN KEY constraints. DML triggers are most useful when the features supported by constraints cannot meet the functional needs of the application.

The following list compares DML triggers with constraints and identifies when DML triggers have benefits over .

- DML triggers can cascade changes through related tables in the database; however, these changes can be executed more efficiently using cascading referential integrity constraints. FOREIGN KEY constraints can validate a column value only with an exact match to a value in another column, unless the REFERENCES clause defines a cascading referential action.
- They can guard against malicious or incorrect INSERT, UPDATE, and DELETE operations and enforce other restrictions that are more complex than those defined with CHECK constraints.

  Unlike CHECK constraints, DML triggers can reference columns in other tables. For example, a trigger can use a SELECT from another table to compare to the inserted or updated data and to perform additional actions, such as modify the data or display a user-defined error message.

- They can evaluate the state of a table before and after a data modification and take actions based on that difference.