SQLSQL Server All (183) Reserved Keywords.

EXPLANATION OF THE KEYWORDS

Md Samiul Hasan

Mob:01538380598

A

SQL SERVER

❖ □ **ADD**
❖ □ **ALL**
❖ □ **ALTER**
❖ □ **AND**
❖ □ **ANY**
❖ □ **AS**
❖ □ **ASC**
❖ □ **AUTHORIZATION**
❖ □ **AVG**

1. ADD

- **Description: The ADD keyword is used with the ALTER TABLE statement to add a new column to an existing table or to add constraints.**
- **Example:**

SQL

```
ALTER TABLE Employees ADD Email VARCHAR(255);
```

This statement adds a new column Email to the Employees table.

2. ALL

- **Description: The ALL keyword is used to compare a value to all values in another set or subquery. It can be used with select statements to return all rows.**
- ***Example:***

SQL

```
SELECT * FROM Employees WHERE Salary > ALL (SELECT Salary FROM
Employees WHERE Department = 'Sales');
```

This query returns all employees whose salary is higher than every employee in the 'Sales' department.

3. ALTER

- ***Description:*** The ALTER keyword is used to modify an existing database object, such as a table. You can add, delete, or modify columns, or change the data type of an existing column.
- ***Example:***

SQL

```
ALTER TABLE Employees ALTER COLUMN LastName VARCHAR(100);
```

This statement changes the data type of the LastName column to VARCHAR(100).

4. AND

- Description: The AND keyword is used to combine two or more conditions in a WHERE clause. All conditions must be true for the record to be included in the result set.
- Example:

SQL

```
SELECT * FROM Employees WHERE Department = 'HR' AND Salary > 50000;
```

This query returns all employees in the HR department who have a salary greater than 50,000.

5. ANY

- Description: The ANY keyword is used to compare a value to any value in a list or subquery. If any of the values meet the condition, the row is included in the result set.
- Example:

SQL

```
SELECT * FROM Employees WHERE Salary > ANY (SELECT Salary FROM
Employees WHERE Department = 'IT');
```

This query returns employees whose salary is higher than any one employee in the 'IT' department.

6. AS

- Description: The AS keyword is used to rename a column or table with an alias in a SQL query, making the output more readable or manageable.
- Example:

SQL

```
SELECT FirstName AS 'First Name', LastName AS 'Last Name' FROM Employees;
```

This query selects the FirstName and LastName columns but displays them with the aliases 'First Name' and 'Last Name'.

7. ASC

- Description: The ASC keyword is used to sort the result set in ascending order. It is the default sorting order if ORDER BY is used without specifying ASC or DESC.
- Example

SQL

```
SELECT * FROM Employees ORDER BY LastName ASC;
```

8. AUTHORIZATION

- Description: The AUTHORIZATION keyword is used to assign privileges or roles to users in the database. It defines who has access to specific database objects.
- Example:

SQL

```
GRANT SELECT ON Employees TO user_name WITH GRANT OPTION;
```

This statement grants the user_name the ability to perform SELECT queries on the Employees table and also to grant this privilege to others.

9. AVG

- Description: The AVG function is used to calculate the average value of a numeric column.
- Example:

  SQL

  ```
  SELECT AVG(Salary) FROM Employees WHERE Department = 'Finance';
  ```

  This query returns the average salary of employees in the 'Finance' department.

B

- BACKUP
- BEGIN
- BETWEEN
- BREAK
- BULK
- BY

1. BACKUP

- Description: The BACKUP keyword is used to create a backup of a database. This ensures that a copy of the database is saved and can be restored if needed.
- Example:

SQL

```
BACKUP DATABASE CompanyDB TO DISK = 'C:\Backups\CompanyDB.bak';
```

This command creates a backup of the CompanyDB database and saves it as a .bak file in the specified directory.

2. BEGIN

- Description: The BEGIN keyword is used to start a transaction or a block of code. It's often paired with COMMIT or ROLLBACK to ensure that a series of SQL commands are executed as a single unit.

- ***Example:***

  SQL

  ```
  BEGIN TRANSACTION;
  UPDATE Employees SET Salary = Salary * 1.10 WHERE Department = 'Sales';
  COMMIT;
  ```

  This block begins a transaction, increases the salary of all employees in the 'Sales' department by 10%, and then commits the transaction, making the changes permanent.

3. BETWEEN

- **Description: The BETWEEN keyword is used to filter the result set within a specific range. It works with numeric, text, and date values.**
- **Example:**

  SQL

  ```
  SELECT * FROM Employees WHERE HireDate BETWEEN '2020-01-01' AND
  '2021-12-31';
  ```

  This query selects all employees who were hired between January 1, 2020, and December 31, 2021.

4. BREAK

- **Description: The BREAK keyword is used to exit a loop or control-of-flow statement prematurely. It is commonly used within WHILE loops to stop execution when a condition is met.**
- ***Example:***

  SQL

  ```
  DECLARE @Counter INT = 0;
  WHILE (@Counter < 10)
  BEGIN
     SET @Counter = @Counter + 1;
     IF @Counter = 5
     BEGIN
        BREAK;
     END
  END
  ```

This script increments a counter in a loop, but it exits the loop when the counter reaches 5.

5. BULK

- **Description: The BULK keyword is used in the BULK INSERT statement to import a large amount of data from a file into a database table.**
- **Example:**

SQL

.

```
BULK INSERT Employees
FROM 'C:\Data\EmployeesData.csv'
WITH (FIELDTERMINATOR = ',', ROWTERMINATOR = '\n');
```

This command imports data from the EmployeesData.csv file into the Employees table. The fields in the file are separated by commas, and each row is terminated by a newline.

6. BY

- **Description: The BY keyword is used with GROUP BY, ORDER BY, and similar clauses to specify how the data should be grouped or ordered.**
- **Example:**

SQL

.

```
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department
ORDER BY NumberOfEmployees DESC;
```

This query groups the employees by department, counts the number of employees in each department, and then orders the results by the number of employees in descending order.

C

- ❖ CASCADE
- ❖ CASE
- ❖ CHECK
- ❖ CHECKPOINT
- ❖ CLOSE

- CLUSTERED
- COALESCE
- COLLATE
- COLUMN
- COMMIT
- COMPUTE
- CONSTRAINT
- CONTAINS
- CONTAINSTABLE
- CONTINUE
- CONVERT
- COUNT
- CREATE
- CROSS
- CURRENT
- CURRENT_DATE
- CURRENT_TIME

- CURRENT_TIMESTAMP
- CURRENT_USER
- CURSOR

1. CASCADE

- Description: The CASCADE option is used with DELETE or UPDATE to automatically apply the operation to related rows in other tables, maintaining referential integrity.
- Example:

  SQL

  .

```
CREATE TABLE Orders (
   OrderID int PRIMARY KEY,
   CustomerID int,
   CONSTRAINT FK_CustomerOrder FOREIGN KEY (CustomerID)
   REFERENCES Customers(CustomerID)
   ON DELETE CASCADE
);
```

If a customer is deleted from the Customers table, the CASCADE option ensures that all orders associated with that customer in the Orders table are also deleted.

2. CASE

- **Description: The CASE keyword allows conditional logic within a SQL query. It works like an if-then-else statement.**
- **Example:**

SQL

```
.
SELECT FirstName, LastName,
       CASE
           WHEN Salary > 50000 THEN 'High'
           WHEN Salary BETWEEN 30000 AND 50000 THEN 'Medium'
           ELSE 'Low'
       END AS SalaryCategory
FROM Employees;
```

This query categorizes employees based on their salary into 'High', 'Medium', or 'Low' categories.

3. CHECK

- **Description: The CHECK constraint is used to enforce a condition on the data that can be inserted into a column.**
- **Example:**

SQL

```
.
CREATE TABLE Employees (
    EmployeeID int PRIMARY KEY,
    FirstName varchar(255),
    LastName varchar(255),
    Age int CHECK (Age >= 18)
);
```

This constraint ensures that only employees aged 18 or older can be added to the Employees table.

4. CHECKPOINT

- **Description: The CHECKPOINT keyword is used to save the current state of the database to disk. It forces a checkpoint, which writes all dirty pages to disk, reducing the time required for recovery.**
- **Example:**

SQL

```
.
CHECKPOINT;
```

This command triggers a checkpoint in the database, which helps in minimizing recovery time after a failure.

5. CLOSE

- **Description: The CLOSE keyword is used to close a cursor after it has been used. This releases the cursor's resources.**
- **Example:**

SQL

```
.
DECLARE EmployeeCursor CURSOR FOR SELECT * FROM Employees;
OPEN EmployeeCursor;
-- Fetch data and process it
CLOSE EmployeeCursor;
DEALLOCATE EmployeeCursor;
```

After processing data with the EmployeeCursor, it is closed to free up resources.

6. CLUSTERED

- **Description: A CLUSTERED index is one where the data rows are stored in order based on the index key. There can be only one clustered index per table because the data rows themselves are sorted.**
- **Example:**

SQL

```
.
CREATE CLUSTERED INDEX IDX_EmployeeID ON Employees(EmployeeID);
```

This creates a clustered index on the EmployeeID column of the Employees table, sorting the table's data rows based on this column.

7. COALESCE

- **Description: The COALESCE function returns the first non-null expression among its arguments.**
- **Example:**

SQL

.

```
SELECT COALESCE(Phone, Mobile, 'No contact info') AS ContactNumber
FROM Employees;
```

This query returns the first non-null contact number from Phone or Mobile. If both are null, it returns 'No contact info'.

8. COLLATE

- **Description:** The COLLATE keyword is used to specify the collation for string data, which defines how data is sorted and compared.
- **Example:**

SQL

.

```
SELECT * FROM Employees WHERE LastName = 'smith' COLLATE
SQL_Latin1_General_CP1_CI_AS;
```

This query finds all employees with the last name 'smith' using a case-insensitive collation.

9. COLUMN

- **Description:** The COLUMN keyword is used to refer to or define a column within a table.
- **Example:**

SQL

.

```
ALTER TABLE Employees ADD COLUMN Email VARCHAR(255);
```

This statement adds a new column named Email to the Employees table.

10. COMMIT

- **Description:** The COMMIT keyword is used to save all changes made during the current transaction to the database.
- **Example:**

SQL

.

```
BEGIN TRANSACTION;
```

```
UPDATE Employees SET Salary = Salary * 1.10 WHERE Department = 'HR';
COMMIT;
```

This transaction increases the salary of HR employees by 10% and commits the changes.

11. COMPUTE

- **Description:** The COMPUTE keyword is used in SQL Server to calculate summary values, such as sums or averages, across groups of data.
- **Example:**

SQL

```
.
SELECT Department, Salary
FROM Employees
ORDER BY Department
COMPUTE SUM(Salary) BY Department;
```

This query calculates the total salary for each department.

12. CONSTRAINT

- **Description:** The CONSTRAINT keyword is used to define rules that limit the type of data that can be stored in a table. Common constraints include PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, and DEFAULT.
- **Example:**

SQL

```
.
CREATE TABLE Employees (
   EmployeeID int PRIMARY KEY,
   LastName varchar(255) NOT NULL,
   Age int CHECK (Age >= 18)
);
```

This statement creates a PRIMARY KEY constraint on EmployeeID and a CHECK constraint on Age.

13. CONTAINS

- **Description: The CONTAINS keyword is used to search for specific words or phrases within a text column, typically in full-text search scenarios.**
- **Example:**

  SQL

  .

  ```
  SELECT * FROM Articles WHERE CONTAINS(Content, 'SQL');
  ```

  This query returns all articles where the Content column contains the word 'SQL'.

14. CONTAINSTABLE

- **Description: The CONTAINSTABLE function returns a table of zero or more rows representing a full-text search match. It's often used to find the relevance ranking for full-text searches.**
- **Example:**

  SQL

  .

  ```
  SELECT A.ArticleID, K.RANK
  FROM Articles A
  JOIN CONTAINSTABLE(Articles, Content, 'SQL') AS K
  ON A.ArticleID = K.[KEY];
  ```

  This query returns articles that contain the word 'SQL', along with their relevance ranking.

15. CONTINUE

- **Description: The CONTINUE keyword is used to skip the current iteration of a loop and move to the next iteration.**
- **Example:**

  SQL

  .

  ```
  DECLARE @Counter INT = 0;
  WHILE @Counter < 10
  BEGIN
     SET @Counter = @Counter + 1;
     IF @Counter = 5 CONTINUE;
     PRINT @Counter;
  ```

```
END;
```

This script skips printing the value when @Counter equals 5 and continues with the next iteration.

16. CONVERT

- Description: The CONVERT function is used to explicitly convert an expression of one data type to another.
- Example:

SQL

.

```
SELECT CONVERT(VARCHAR, HireDate, 103) AS HireDateFormatted
FROM Employees;
```

This query converts the HireDate column to a string in the format dd/mm/yyyy.

17. COUNT

- Description: The COUNT function returns the number of rows that match a specified condition.
- Example:

SQL

.

```
SELECT COUNT(*) AS NumberOfEmployees FROM Employees WHERE
Department = 'Sales';
```

This query counts the number of employees in the 'Sales' department.

18. CREATE

- Description: The CREATE keyword is used to create a new database object, such as a table, index, view, or stored procedure.
- Example:

SQL

.

```
CREATE TABLE Employees (
    EmployeeID int PRIMARY KEY,
    FirstName varchar(255),
    LastName varchar(255)
);
```

This statement creates a new table named Employees with three columns.

19. CROSS

- Description: The CROSS keyword is used in the CROSS JOIN clause, which produces a Cartesian product of two tables. It combines each row of the first table with every row of the second table.
- Example:

SQL

.

```
SELECT A.FirstName, B.DepartmentName
FROM Employees A
CROSS JOIN Departments B;
```

This query returns every possible combination of employees and departments.

20. CURRENT

- Description: The CURRENT keyword is used to refer to the current value or context in specific SQL functions or commands.
- Example:

SQL

.

```
SELECT CURRENT_USER;
```

This query returns the name of the currently logged-in user.

21. CURRENT_DATE

- Description: The CURRENT_DATE keyword returns the current date. It does not include the time part.
- Example:

SQL

.

```
SELECT CURRENT_DATE AS TodayDate;
```

This query returns the current date.

22. CURRENT_TIME

- **Description: The CURRENT_TIME keyword returns the current time. It does not include the date part.**
- **Example:**

SQL

.

```
SELECT CURRENT_TIME AS NowTime;
```

This query returns the current time.

23. CURRENT_TIMESTAMP

- **Description: The CURRENT_TIMESTAMP keyword returns the current date and time.**
- **Example:**

SQL

.

```
SELECT CURRENT_TIMESTAMP AS Now;
```

This query returns the current date and time.

24. CURRENT_USER

- **Description: The CURRENT_USER keyword returns the name of the current user in the database.**
- **Example:**

SQL

.

```
SELECT CURRENT_USER;
```

This query returns the username of the person currently logged into the database.

25. CURSOR

- **Description: The CURSOR keyword is used to declare a database cursor, which allows you to retrieve, manipulate, and navigate through the result set row by row.**
- **Example:**

SQL

```
.
DECLARE EmployeeCursor CURSOR FOR SELECT FirstName, LastName FROM
Employees;
OPEN EmployeeCursor;
FETCH NEXT FROM EmployeeCursor INTO @FirstName, @LastName;
WHILE @@FETCH_STATUS = 0
BEGIN
   PRINT @FirstName + ' ' + @LastName;
   FETCH NEXT FROM EmployeeCursor INTO @FirstName, @LastName;
END;
CLOSE EmployeeCursor;
DEALLOCATE EmployeeCursor;
```

D

- **DATABASE**
- **DBCC**
- **DEALLOCATE**
- **DECLARE**
- **DEFAULT**
- **DELETE**
- **DENY**
- **DESC**
- **DISK**
- **DISTINCT**
- **DISTRIBUTED**
- **DOUBLE**
- **DROP**
- **DUMP**
-

1. DATABASE

- **Description: The DATABASE keyword is used to create, alter, or delete a database. It defines a collection of related data.**
- ***Example:***

  SQL

  .

  ```
  CREATE DATABASE MyDatabase;
  ```

  This command creates a new database named MyDatabase.

2. DBCC

- **Description: DBCC stands for Database Console Commands. It's used for various database maintenance tasks, such as checking the integrity of the database or shrinking it.**
- **Example:**

  SQL

  .

  ```
  DBCC CHECKDB ('MyDatabase');
  ```

  This command checks the integrity of the MyDatabase database.

3. DEALLOCATE

- **Description: The DEALLOCATE keyword is used to release resources associated with a cursor. It's typically used after a cursor is closed.**
- **Example:**

  SQL

  .

  ```
  DEALLOCATE EmployeeCursor;
  ```

  This command releases the resources associated with the cursor named EmployeeCursor.

4. DECLARE

- **Description: The DECLARE keyword is used to create variables, cursors, or conditions. It sets up the objects that will be used in the subsequent SQL statements.**
- **Example:**

SQL

```
.
DECLARE @EmployeeID int;
SET @EmployeeID = 1;
```

This statement declares a variable @EmployeeID and assigns it a value of 1.

5. DEFAULT

- **Description:** The DEFAULT keyword is used to provide a default value for a column when a new row is inserted without specifying a value for that column.
- **Example:**

SQL

```
.
CREATE TABLE Employees (
   EmployeeID int PRIMARY KEY,
   FirstName varchar(255) DEFAULT 'John',
   LastName varchar(255)
);
```

This statement creates a table with a default value of 'John' for the FirstName column if no value is provided during insertion.

6. DELETE

- **Description:** The DELETE keyword is used to remove one or more rows from a table based on a specified condition.
- **Example:**

SQL

```
.
DELETE FROM Employees WHERE EmployeeID = 1;
```

This command deletes the employee with an EmployeeID of 1 from the Employees table.

7. DENY

- **Description: The DENY keyword is used to explicitly prevent a user from accessing certain database objects or performing specific actions.**
- **Example:**

  SQL

  .

  ```
  DENY SELECT ON Employees TO user_name;
  ```

  This command denies the SELECT permission on the Employees table to user_name.

8. DESC

- **Description: The DESC keyword is used to sort the result set in descending order. It is often used with the ORDER BY clause.**
- **Example:**

  SQL

  .

  ```
  SELECT * FROM Employees ORDER BY Salary DESC;
  ```

  This query returns all employees sorted by their salary in descending order.

9. DISK

- **Description: The DISK keyword is used in commands related to disk storage, often in the context of backup and restore operations.**
- **Example:**

  SQL

  .

  ```
  BACKUP DATABASE MyDatabase TO DISK = 'C:\Backups\MyDatabase.bak';
  ```

  This command backs up the MyDatabase database to a file on disk.

10. DISTINCT

- **Description: The DISTINCT keyword is used to remove duplicate rows from the result set.**
- **Example:**

SQL

.

```
SELECT DISTINCT Department FROM Employees;
```

This query returns a list of unique departments from the Employees table.

11. DISTRIBUTED

- **Description: The DISTRIBUTED keyword is used in the context of distributed queries or distributed databases, where data is spread across multiple locations.**
- **Example:**

SQL

.

```
SELECT * FROM [LinkedServer].[Database].[Schema].[Table];
```

This query retrieves data from a table on a remote or linked server, demonstrating a distributed query.

12. DOUBLE

- **Description: The DOUBLE keyword is used to define a column with a double-precision floating-point data type, commonly used for numeric values with higher precision.**
- **Example:**

SQL

.

```
CREATE TABLE Products (
   ProductID int PRIMARY KEY,
   Price DOUBLE
);
```

This statement creates a table with a Price column that uses the DOUBLE data type for storing numeric values.

13. DROP

- **Description: The DROP keyword is used to remove a database object, such as a table, view, or index. This operation is irreversible.**

- **Example:**

  SQL
  .
  ```
  DROP TABLE Employees;
  ```

  This command deletes the Employees table and all of its data.

14. DUMP

- **Description: The DUMP keyword is used in SQL Server to generate a binary or text file containing database information, often used for debugging or logging purposes.**
- **Example:**

  SQL
  .
  ```
  DUMP DATABASE MyDatabase TO 'C:\Backups\MyDatabaseDump.dmp';
  ```

  This command creates a dump file of the MyDatabase database.

E

- ❖ **ELSE**
- ❖ **END**
- ❖ **ERRLVL**
- ❖ **ESCAPE**
- ❖ **EXCEPT**
- ❖ **EXEC**
- ❖ **EXECUTE**
- ❖ **EXISTS**
- ❖ **EXIT**
- ❖ **EXTERNAL**

1. ELSE

- **Description: The ELSE keyword is used in conjunction with CASE statements or IF conditions to define what should happen when none of the WHEN conditions are met.**
- **Example:**

  SQL

```
.
SELECT FirstName, LastName,
       CASE
           WHEN Salary > 50000 THEN 'High'
           WHEN Salary BETWEEN 30000 AND 50000 THEN 'Medium'
           ELSE 'Low'
       END AS SalaryCategory
FROM Employees;
```

This query categorizes employees based on their salary into 'High', 'Medium', or 'Low' if none of the previous conditions apply.

2. END

- Description: The END keyword is used to signify the end of a CASE statement, a block of code within BEGIN...END, or a TRY...CATCH block.
- Example:

SQL

```
.
BEGIN
    -- Some SQL statements
END;
```

This marks the end of a block of SQL statements enclosed by BEGIN.

3. ERRLVL

- Description: The ERRLVL keyword is used with RAISEERROR in SQL Server to specify the severity level of an error.
- Example:

SQL

```
.
RAISERROR ('An error occurred', 16, 1);
```

This raises an error message with a severity level of 16. The ERRLVL parameter is used to determine the seriousness of the error.

4. ESCAPE

- **Description: The ESCAPE keyword is used with LIKE to define a character that will be used to escape wildcard characters in pattern matching.**
- **Example:**

SQL
.

```
SELECT * FROM Products WHERE ProductName LIKE '100\% Complete'
ESCAPE '\';
```

This query searches for product names that contain the string '100% Complete', using \ as the escape character for the % wildcard.

5. EXCEPT

- **Description: The EXCEPT keyword is used to return rows from the first query that are not present in the second query. It's similar to MINUS in other SQL dialects.**
- **Example:**

SQL
.

```
SELECT ProductID FROM Products
EXCEPT
SELECT ProductID FROM Orders;
```

This query returns all ProductID values that are in the Products table but not in the Orders table.

6. EXEC

- **Description: The EXEC keyword is used to execute a stored procedure or a dynamic SQL statement.**
- **Example:**

SQL
.

```
EXEC sp_EmployeeDetails @EmployeeID = 1;
```

This command executes the stored procedure sp_EmployeeDetails with a parameter @EmployeeID set to 1.

7. EXECUTE

- **Description: The EXECUTE keyword is similar to EXEC and is used to run a stored procedure or a dynamic SQL statement.**
- **Example:**

  SQL

  .

  ```
  EXECUTE sp_GetEmployeeInfo @EmployeeID = 2;
  ```

  This command executes the stored procedure sp_GetEmployeeInfo with the parameter @EmployeeID set to 2.

8. EXISTS

- **Description: The EXISTS keyword is used in a subquery to check if any rows are returned. It returns TRUE if the subquery returns one or more rows, otherwise FALSE.**
- **Example:**

  SQL

  .

  ```
  SELECT FirstName, LastName
  FROM Employees
  WHERE EXISTS (SELECT * FROM Orders WHERE Orders.EmployeeID =
  Employees.EmployeeID);
  ```

  This query retrieves employees who have at least one associated order.

9. EXIT

- **Description: The EXIT keyword is used to exit from a loop or a block of code. It is commonly used in procedural SQL.**
- **Example:**

  SQL

  .

  ```
  DECLARE @Counter INT = 0;
  WHILE @Counter < 10
  BEGIN
     SET @Counter = @Counter + 1;
     IF @Counter = 5 EXIT;
  ```

```
    PRINT @Counter;
END;
```

This script exits the loop when the counter reaches 5.

10. EXTERNAL

- **Description:** The **EXTERNAL** keyword is used in certain contexts to specify that an operation involves external data or objects, such as external tables or files.
- **Example:**

SQL

.

```
CREATE EXTERNAL DATA SOURCE MyDataSource
WITH (
    TYPE = HADOOP,
    LOCATION = 'hdfs://myhadoopcluster'
);
```

F

- FETCH
- FILE
- FILLFACTOR
- FOR
- FOREIGN
- FREETEXT
- FREETEXTTABLE
- FROM
- FULL
- FUNCTION

1. FETCH

- **Description:** The **FETCH** keyword is used with cursors to retrieve the next row from the result set into a variable.
- **Example:**

SQL

.

```
DECLARE @FirstName NVARCHAR(50);
DECLARE @LastName NVARCHAR(50);
DECLARE EmployeeCursor CURSOR FOR
    SELECT FirstName, LastName FROM Employees;
OPEN EmployeeCursor;
FETCH NEXT FROM EmployeeCursor INTO @FirstName, @LastName;
WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT @FirstName + ' ' + @LastName;
    FETCH NEXT FROM EmployeeCursor INTO @FirstName, @LastName;
END;
CLOSE EmployeeCursor;
DEALLOCATE EmployeeCursor;
```

This script declares a cursor, fetches the first row of data, and processes each row in a loop.

2. FILE

- **Description:** The FILE keyword is used in certain SQL Server commands related to file management, such as specifying file locations for database backups or restores.
- **Example:**

SQL

.

```
BACKUP DATABASE MyDatabase TO DISK = 'C:\Backups\MyDatabase.bak';
```

This command backs up MyDatabase to a specified file on disk.

3. FILLFACTOR

- **Description:** The FILLFACTOR keyword is used in index creation or alteration to specify the percentage of space on each leaf-level page to be filled with data. It helps manage page splits and performance.
- **Example:**

SQL

.

```
CREATE INDEX IX_Employee_LastName ON Employees(LastName)
WITH (FILLFACTOR = 80);
```

This statement creates an index with a fill factor of 80%, meaning each page will be filled to 80% capacity.

4. FOR

- Description: The `FOR` keyword is used in various SQL constructs to define actions or behaviors, such as specifying cursors or triggers.
- Example:

  SQL

  ```
  .
  DECLARE EmployeeCursor CURSOR FOR
      SELECT FirstName, LastName FROM Employees;
  ```

  This declares a cursor `EmployeeCursor` for selecting employee names.

  SQL

  ```
  .
  CREATE TRIGGER trgAfterInsert
  ON Employees
  FOR INSERT
  AS
  BEGIN
      PRINT 'A new record was inserted into Employees';
  END;
  ```

  This creates a trigger that runs after an `INSERT` operation on the `Employees` table.

5. FOREIGN

- Description: The `FOREIGN` keyword is used to define a foreign key constraint, which establishes a relationship between columns in different tables.
- Example:

  SQL

  ```
  .
  CREATE TABLE Orders (
      OrderID int PRIMARY KEY,
      CustomerID int,
      CONSTRAINT FK_Customer FOREIGN KEY (CustomerID)
      REFERENCES Customers(CustomerID)
  );
  ```

This statement creates a foreign key constraint on CustomerID, linking it to the Customers table.

6. FREETEXT

- **Description:** The FREETEXT keyword is used in full-text search to search for words or phrases in text columns, with more flexibility than LIKE.
- **<u>Example:</u>**

SQL

```
.
SELECT * FROM Articles
WHERE FREETEXT(Content, 'database optimization');
```

This query searches for articles containing the words 'database' and 'optimization' in the Content column.

7. FREETEXTTABLE

- **Description:** The FREETEXTTABLE function returns a table of rows with relevance rankings based on a full-text search query.
- **<u>Example:</u>**

SQL

```
.
SELECT ArticleID, RANK
FROM FREETEXTTABLE(Articles, Content, 'database optimization');
```

This query returns article IDs and relevance rankings for articles matching the full-text search 'database optimization'.

8. FROM

- **Description:** The FROM keyword specifies the tables or views from which to retrieve data in a SELECT statement.
- **<u>Example:</u>**

SQL

```
.
SELECT FirstName, LastName FROM Employees;
```

This command selects FirstName and LastName columns from the Employees table.

9. FULL

- **Description: The FULL keyword is used in the FULL JOIN clause to include all rows from both joined tables, with NULL values where there is no match.**
- **Example:**

SQL

.

```
SELECT A.FirstName, B.DepartmentName
FROM Employees A
FULL JOIN Departments B ON A.DepartmentID = B.DepartmentID;
```

This query returns all employees and departments, including rows with no match in either table.

10. FUNCTION

- **Description: The FUNCTION keyword is used to define a user-defined function, which can return a single value or a table.**
- **Example:**

SQL

.

```
CREATE FUNCTION GetEmployeeFullName(@EmployeeID int)
RETURNS varchar(255)
AS
BEGIN
   DECLARE @FullName varchar(255);
   SELECT @FullName = FirstName + ' ' + LastName
   FROM Employees
   WHERE EmployeeID = @EmployeeID;
   RETURN @FullName;
END;
```

This statement creates a function GetEmployeeFullName that concatenates the first and last names of an employee based on their ID.

G

- GOTO
- GRANT
- GROUP
- HAVING
- HOLDLOCK

1. GOTO

- **Description: The GOTO keyword is used to jump to a specific part of the code, usually within a block of code or a stored procedure.**
- **Example:**

  SQL

  ```
  GOTO EndLabel;

  -- Some code here

  EndLabel:
  PRINT 'Reached the end of the block.';
  ```

  This command jumps to the EndLabel part of the code.

2. GRANT

- **Description: The GRANT keyword is used to give specific permissions to users or roles, such as the ability to read or modify data.**
- **Example:**

  SQL

  ```
  GRANT SELECT ON Employees TO user_name;
  ```

  This command allows the user user_name to read data from the Employees table.

3. GROUP

- **Description: The GROUP keyword is used with GROUP BY to aggregate data into groups based on one or more columns.**
- **Example:**

SQL

```
.
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department;
```

This query counts the number of employees in each department by grouping them based on the Department column.

4. HAVING

- **Description:** The HAVING keyword is used to filter groups created by GROUP BY. It works similarly to WHERE, but is applied to aggregated data.
- ***Example:***

SQL

```
SELECT Department, COUNT(*) AS NumberOfEmployees
FROM Employees
GROUP BY Department
HAVING COUNT(*) > 5;
```

This query returns departments with more than 5 employees.

5. HOLDLOCK

- **Description:** The HOLDLOCK keyword is used to specify a locking behavior that holds a lock until the end of a transaction. It ensures that no other transactions can modify the data until the current transaction is completed.
- ***Example:***

SQL

```
BEGIN TRANSACTION;
SELECT * FROM Employees WITH (HOLDLOCK);
-- Some processing here
COMMIT TRANSACTION;
```

This query locks the Employees table until the transaction is complete, preventing others from making changes during that time.

I

- **IDENTITY**
- **IDENTITY_INSERT**
- **IDENTITYCOL**
- **IF**
- **IN**
- **INDEX**
- **INNER**
- **INSERT**
- **INTERSECT**
- **INTO**
- **IS**

1. IDENTITY

- **Description: The IDENTITY keyword is used to create an auto-incrementing column in a table, often used for primary keys.**
- ***Example:***

SQL

.

```
CREATE TABLE Employees (
    EmployeeID int IDENTITY(1,1) PRIMARY KEY,
    FirstName varchar(50),
    LastName varchar(50)
);
```

In this example, EmployeeID will automatically increment by 1 for each new row.

2. IDENTITY_INSERT

- **Description: The IDENTITY_INSERT keyword allows you to manually insert values into an identity column, overriding its auto-increment behavior.**
- **Example:**

SQL

.

```
SET IDENTITY_INSERT Employees ON;
INSERT INTO Employees (EmployeeID, FirstName, LastName) VALUES (1,
'John', 'Doe');
SET IDENTITY_INSERT Employees OFF;
```

This allows you to insert a specific EmployeeID value into the Employees table.

3. IDENTITYCOL

- **Description: The IDENTITYCOL keyword is used to refer to an identity column in SQL Server, typically within system functions.**
- **Example:**

SQL

```
.
SELECT IDENT_CURRENT('Employees') AS LastIdentity;
```

This returns the last identity value inserted into the Employees table.

4. IF

- **Description: The IF keyword is used for conditional logic to execute certain code only if a specified condition is true.**
- **Example:**

SQL

```
.
IF EXISTS (SELECT * FROM Employees WHERE EmployeeID = 1)
BEGIN
   PRINT 'Employee with ID 1 exists.';
END
ELSE
BEGIN
   PRINT 'Employee with ID 1 does not exist.';
END
```

This script checks if an employee with ID 1 exists and prints a message accordingly.

5. IN

- **Description: The IN keyword is used to check if a value is within a specified set of values.**
- **Example:**

SQL

```
.
```

```
SELECT * FROM Employees
WHERE Department IN ('Sales', 'Marketing');
```

This query retrieves employees who are in either the Sales or Marketing department.

6. INDEX

- **Description: The INDEX keyword is used to create or manage indexes, which improve the speed of data retrieval operations on a table.**
- **Example:**

SQL

.

```
CREATE INDEX IX_LastName ON Employees(LastName);
```

This creates an index on the LastName column to speed up searches involving this column.

7. INNER

- **Description: The INNER keyword is used in INNER JOIN to combine rows from two tables based on a related column, returning only the rows with matching values in both tables.**
- **Example:**

SQL

.

```
SELECT Employees.FirstName, Departments.DepartmentName
FROM Employees
INNER JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

This query retrieves employee names and their corresponding department names, showing only those with matches in both tables.

8. INSERT

- **Description: The INSERT keyword is used to add new rows of data into a table.**
- **Example:**

SQL

.

```
INSERT INTO Employees (FirstName, LastName, DepartmentID)
VALUES ('Jane', 'Smith', 2);
```

This command inserts a new employee with the first name 'Jane', last name 'Smith', and department ID 2 into the Employees table.

9. INTERSECT

- Description: The INTERSECT keyword is used to return the common rows from two queries, similar to finding the intersection of two sets.
- Example:

SQL

.

```
SELECT ProductID FROM Orders
INTERSECT
SELECT ProductID FROM Inventory;
```

This query returns the ProductID values that are present in both the Orders and Inventory tables.

10. INTO

- Description: The INTO keyword is used to select data and insert it into a new table or variable.
- Example:

SQL

.

```
SELECT * INTO BackupEmployees
FROM Employees;
```

This creates a new table BackupEmployees with the same structure and data as the Employees table.

11. IS

- Description: The IS keyword is used in conditions to test for null values or to check specific conditions.
- Example:

SQL

.

```
SELECT FirstName, LastName
FROM Employees
WHERE ManagerID IS NULL;
```

J

- ❖ JOIN

JOIN

- Description: The JOIN keyword is used to combine rows from two or more tables based on a related column between them. It allows you to query data from multiple tables in a single query.
- Types of Joins:
  1. INNER JOIN: Returns only the rows where there is a match in both tables.
  2. LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table and matched rows from the right table. If there is no match, the result is NULL for columns from the right table.
  3. RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table and matched rows from the left table. If there is no match, the result is NULL for columns from the left table.
  4. FULL JOIN (or FULL OUTER JOIN): Returns all rows when there is a match in one of the tables. It combines the results of both LEFT JOIN and RIGHT JOIN.
  5. CROSS JOIN: Returns the Cartesian product of the two tables, i.e., all possible combinations of rows from the two tables.
- Examples:
  1. INNER JOIN:

     SQL

     .

     ```
     SELECT Employees.FirstName, Departments.DepartmentName
     FROM Employees
     INNER JOIN Departments ON Employees.DepartmentID =
     Departments.DepartmentID;
     ```

This query returns a list of employee names and their respective department names, showing only employees who have a matching department.

2. LEFT JOIN:

SQL

.

```
SELECT Employees.FirstName, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

This query returns all employees and their department names. Employees without a department will have NULL for the department name.

3. RIGHT JOIN:

SQL

.

```
SELECT Employees.FirstName, Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

This query returns all departments and the names of employees in those departments. Departments with no employees will have NULL for the employee name.

4. FULL JOIN:

SQL

.

```
SELECT Employees.FirstName, Departments.DepartmentName
FROM Employees
FULL JOIN Departments ON Employees.DepartmentID =
Departments.DepartmentID;
```

This query returns all employees and all departments, with NULL in columns where there is no match in one of the tables.

5. CROSS JOIN:

SQL

```
.
SELECT Employees.FirstName, Departments.DepartmentName
FROM Employees
CROSS JOIN Departments;
```

This query returns all possible combinations of employee names and department names.

K

- ❖ **KEY**
- ❖ **KILL**

1. KEY

- **Description: The KEY keyword is used in the context of creating or modifying indexes, primary keys, or foreign keys in a table. It defines columns that uniquely identify rows or establish relationships between tables.**
- ***Example:***

a. Creating a Primary Key:

SQL

```
.
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50)
);
```

Here, EmployeeID is defined as the primary key, which ensures that each EmployeeID is unique and not null.

b. Creating a Foreign Key:

SQL.

```
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   EmployeeID INT,
   FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
```

In this example, EmployeeID in the Orders table is defined as a foreign key that references the EmployeeID column in the Employees table.

c. Creating an Index:

SQL

```
.
CREATE INDEX idx_LastName ON Employees(LastName);
```

This creates an index on the LastName column of the Employees table to improve query performance.

2. KILL

- Description: The KILL command is used to terminate a specific session or process in SQL Server. It is often used to stop a runaway query or session that is blocking resources or causing issues.
- Example:

a. Killing a Specific Session:

SQL

```
.
KILL 57;
```

This command terminates the session with the ID 57. The session ID can be found using the sp_who or sp_who2 system stored procedures.

b. Finding and Killing Blocking Sessions:

SQL

```
.
-- Find blocking sessions
SELECT blocking_session, session_id, wait_type, wait_time
FROM sys.dm_exec_requests
```

```
WHERE blocking_session <> 0;

-- Kill the blocking session (example session ID 58)
KILL 58;
```

L

- ❖ LEFT
- ❖ LIKE
- ❖ LINENO
- ❖ LOAD

1. LEFT

- **Description: The LEFT function is used to extract a specified number of characters from the beginning (left side) of a string.**
- **Example:**

  SQL

  .

  ```
  SELECT LEFT(FirstName, 3) AS Initials
  FROM Employees;
  ```

  This query extracts the first three characters from the FirstName column for each employee, returning them as Initials.

2. LIKE

- **Description: The LIKE keyword is used in WHERE clauses to search for a specified pattern in a column. It is often used with wildcard characters:**
  - ○ **% represents zero or more characters.**
  - ○ **_ represents a single character.**
- **Example:**

  SQL

  .

  ```
  SELECT * FROM Employees
  WHERE LastName LIKE 'S%';
  ```

This query retrieves all employees whose last names start with the letter 'S'.

SQL

```
.
SELECT * FROM Products
WHERE ProductName LIKE '_x%';
```

This query retrieves products where the second character is 'x', and it can be followed by any number of characters.

3. LINENO

- **Description:** The LINENO function returns the current line number of the executing statement in a batch or stored procedure. This function is often used for debugging purposes.
- **Example:**

SQL

```
.
PRINT 'Current line number is: ' + CAST(LINENO() AS VARCHAR);
```

This command prints the current line number of the statement being executed.

Note: LINENO is more commonly used in some procedural SQL environments like T-SQL. Its behavior and availability might vary depending on the SQL dialect and version.

4. LOAD

- **Description:** The LOAD keyword is used to import data from an external source into a table or database. It's often used in conjunction with commands or utilities that handle data files.
- ***Example:***

SQL

```
.
LOAD DATA INFILE 'path/to/file.csv'
INTO TABLE Employees
FIELDS TERMINATED BY ','
```

```
LINES TERMINATED BY '\n'
(FirstName, LastName, DepartmentID);
```

This example loads data from a CSV file into the Employees table, specifying delimiters for fields and lines.

M

- ❖ **MERGE**

MERGE

- **Description: The MERGE statement is used to perform an INSERT, UPDATE, or DELETE operation in a single statement based on the results of a JOIN between a target table and a source table or dataset. It is useful for synchronizing tables and applying multiple operations in a single transaction.**
- **Syntax:**

  SQL

  .

```
MERGE INTO TargetTable AS target
USING SourceTable AS source
ON target.KeyColumn = source.KeyColumn
WHEN MATCHED THEN
   UPDATE SET target.Column1 = source.Column1, target.Column2 =
source.Column2
WHEN NOT MATCHED BY TARGET THEN
   INSERT (Column1, Column2) VALUES (source.Column1, source.Column2)
WHEN NOT MATCHED BY SOURCE THEN
   DELETE;
```

- **Examples:**

1. Basic Example:

SQL

.

```
MERGE INTO Employees AS target
USING StagingEmployees AS source
ON target.EmployeeID = source.EmployeeID
```

```
WHEN MATCHED THEN
   UPDATE SET target.FirstName = source.FirstName, target.LastName =
source.LastName
WHEN NOT MATCHED BY TARGET THEN
   INSERT (EmployeeID, FirstName, LastName)
   VALUES (source.EmployeeID, source.FirstName, source.LastName)
WHEN NOT MATCHED BY SOURCE THEN
   DELETE;
```

This statement synchronizes the Employees table with data from StagingEmployees:

- Updates existing employees with new data.
- Inserts new employees that are not in the Employees table.
- Deletes employees that are no longer in the StagingEmployees table.

2. Example with Conditions:

SQL

.

```
MERGE INTO Orders AS target
USING NewOrders AS source
ON target.OrderID = source.OrderID
WHEN MATCHED AND source.OrderStatus = 'Shipped' THEN
   UPDATE SET target.ShippingDate = source.ShippingDate
WHEN NOT MATCHED BY TARGET AND source.OrderStatus = 'Pending' THEN
   INSERT (OrderID, OrderDate, OrderStatus)
   VALUES (source.OrderID, source.OrderDate, source.OrderStatus)
WHEN NOT MATCHED BY SOURCE THEN
   DELETE;
```

N

- NATIONAL
- NOCHECK
- NONCLUSTERED
- NOT
- NULL
- NULLIF

1. NATIONAL

- **Description: The NATIONAL keyword is used in some SQL dialects to specify national character data types. For example, NATIONAL CHAR or NATIONAL VARCHAR in certain SQL implementations are used to store Unicode characters.**
- **Example:**

SQL

.

```
CREATE TABLE Customers (
   CustomerID INT PRIMARY KEY,
   CustomerName NATIONAL VARCHAR(100)
);
```

In this example, CustomerName is defined to store Unicode characters, allowing for a broader range of international characters.

2. NOCHECK

- **Description: The NOCHECK keyword is used in SQL Server to specify that constraints (like foreign key constraints) should not be checked during certain operations. It is often used in conjunction with ALTER TABLE or CHECK CONSTRAINT commands.**
- **Example:**

SQL

.

```
ALTER TABLE Orders NOCHECK CONSTRAINT FK_CustomerOrder;
```

This command disables the check for the FK_CustomerOrder foreign key constraint on the Orders table.

3. NONCLUSTERED

- **Description: The NONCLUSTERED keyword is used to define a type of index that is separate from the data table, allowing for quicker retrieval of rows based on the index key. Unlike a clustered index, a nonclustered index does not alter the physical order of data in the table.**
- **Example:**

SQL

.

```
CREATE NONCLUSTERED INDEX idx_LastName
ON Employees(LastName);
```

This creates a nonclustered index on the LastName column of the Employees table, which improves query performance when searching by last name.

4. NOT

- **Description: The NOT keyword is used in logical expressions to negate a condition. It returns TRUE if the condition is FALSE, and FALSE if the condition is TRUE.**
- **Example:**

SQL

.

```
SELECT * FROM Employees
WHERE NOT Department = 'HR';
```

This query retrieves all employees who are not in the 'HR' department.

SQL

.

```
SELECT * FROM Orders
WHERE NOT (OrderDate BETWEEN '2023-01-01' AND '2023-12-31');
```

This query retrieves orders placed outside the year 2023.

5. NULL

- **Description: The NULL keyword represents a missing or undefined value in a column. It is different from an empty string or zero, and it signifies the absence of data.**
- **Example:**

SQL

.

```
SELECT * FROM Employees
WHERE MiddleName IS NULL;
```

This query retrieves all employees who do not have a middle name (i.e., the MiddleName column contains NULL).

6. NULLIF

- **Description: The NULLIF function returns NULL if two expressions are equal; otherwise, it returns the first expression. It is used to avoid division by zero or handle cases where two values should be considered as NULL.**
- **Example:**

SQL

.

```
SELECT NULLIF(Amount, 0) AS SafeAmount
FROM Transactions;
```

This query returns NULL if Amount is zero, otherwise, it returns the value of Amount.

SQL

.

```
SELECT ProductID, NULLIF(Discount, 0) AS DiscountValue
FROM Products;
```

O

- OF
- OFF
- OFFSETS
- ON
- OPEN
- OPENDATASOURCE
- OPENQUERY
- OPENROWSET
- OPENXML
- OPTION
- OR
- ORDER
- OUTER
- OVER

1. OF

- **Description: The OF keyword is often used in the context of defining or modifying constraints, such as in ALTER TABLE commands, but its usage is specific to certain SQL dialects or versions.**
- **Example:**

  SQL

  .

  ```
  ALTER TABLE Orders
  ADD CONSTRAINT FK_CustomerOrder
  FOREIGN KEY (CustomerID)
  REFERENCES Customers(CustomerID);
  ```

  Here, OF is not directly used, but constraints like FOREIGN KEY are defined in a similar context.

2. OFF

- **Description: The OFF keyword is used to disable certain database features or options, such as constraints, triggers, or specific configurations.**
- **Example:**

  SQL

  .

  ```
  ALTER TABLE Orders NOCHECK CONSTRAINT FK_CustomerOrder;
  ```

  The OFF concept is implied here by NOCHECK, which disables the constraint checks.

3. OFFSETS

- **Description: The OFFSETS keyword is used in SQL Server to specify the starting point and size of data returned from a query. It is used in conjunction with the OFFSET and FETCH clauses for paging results.**
- **Example:**

  SQL

  .

  ```
  SELECT *
  FROM Employees
  ORDER BY EmployeeID
  ```

```
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY;
```

This query skips the first 10 rows and retrieves the next 10 rows from the Employees table.

4. ON

- **Description:** The ON keyword is used to specify conditions in various SQL operations, including JOIN clauses, constraints, and triggers.
- **Example:**

SQL

.

```
SELECT e.FirstName, d.DepartmentName
FROM Employees e
INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

This query joins the Employees table with the Departments table based on the DepartmentID column.

SQL

.

```
CREATE TRIGGER trg_UpdateSalary
ON Employees
AFTER UPDATE
AS
BEGIN
    PRINT 'Salary has been updated';
END;
```

This creates a trigger that executes after an update on the Employees table.

5. OPEN

- **Description:** The OPEN keyword is used to open a cursor in SQL Server, allowing it to start processing rows.
- **Example:**

SQL

.

```
DECLARE cursor_example CURSOR FOR
SELECT FirstName, LastName
FROM Employees;

OPEN cursor_example;
```

This declares and opens a cursor to iterate over employee records.

6. OPENDATASOURCE

- **Description:** The OPENDATASOURCE function is used to access data from a remote data source without setting up a linked server.
- **Example:**

SQL

.

```
SELECT *
FROM OPENDATASOURCE('SQLNCLI',
   'Data Source=RemoteServer;Integrated
Security=SSPI;').DatabaseName.SchemaName.TableName;
```

This query retrieves data from a remote server using the OPENDATASOURCE function.

7. OPENQUERY

- **Description:** The OPENQUERY function executes a pass-through query on a linked server and returns the result.
- **Example:**

SQL

.

```
SELECT *
FROM OPENQUERY(LinkedServerName, 'SELECT * FROM RemoteTable');
```

This query executes the SQL query on a linked server and returns the result.

8. OPENROWSET

- **Description:** The OPENROWSET function allows for ad-hoc access to remote data sources by querying them directly from SQL Server.
- **Example:**

SQL

.

```
SELECT *
FROM OPENROWSET('SQLNCLI',
   'Server=RemoteServer;Trusted_Connection=yes;',
   'SELECT * FROM RemoteDatabase.dbo.RemoteTable');
```

This query accesses data from a remote server and retrieves results from a specific table.

9. OPENXML

- **Description: The OPENXML function parses XML data and returns it as a rowset that can be queried using SQL.**
- **Example:**

SQL

.

```
DECLARE @xml XML;
SET @xml = '<Employees><Employee ID="1"><Name>John</Name></Employee></Employees>';

SELECT *
FROM OPENXML(@xml, '/Employees/Employee', 2)
WITH (ID INT '@ID', Name NVARCHAR(50) 'Name');
```

This parses the XML data and returns it as a tabular result.

10. OPTION

- **Description: The OPTION keyword is used to specify query hints that can influence the execution plan for a query.**
- **Example:**

SQL

.

```
SELECT *
FROM Orders
WHERE OrderDate > '2023-01-01'
OPTION (OPTIMIZE FOR UNKNOWN);
```

This query uses the OPTION keyword to specify the OPTIMIZE FOR UNKNOWN hint, which helps SQL Server optimize the query execution.

11. OR

- **Description: The OR keyword is used in conditional statements to combine multiple conditions. It returns TRUE if at least one of the conditions is TRUE.**
- **Example:**

  SQL

  .

  ```
  SELECT * FROM Employees
  WHERE Department = 'HR' OR Department = 'IT';
  ```

  This query retrieves employees who are in either the 'HR' or 'IT' department.

12. ORDER

- **Description: The ORDER keyword is used in conjunction with BY to sort the results of a query based on one or more columns.**
- **Example:**

  SQL

  .

  ```
  SELECT * FROM Employees
  ORDER BY LastName ASC, FirstName DESC;
  ```

  This query sorts the employees by last name in ascending order and first name in descending order

1. OUTER

- **Description: The OUTER keyword is used in SQL in combination with JOIN to create an OUTER JOIN, which is a type of join that returns all rows from one table and the matching rows from the other table. If there is no match, the result will contain NULL values for columns from the table without a match. There are three types of OUTER JOINs:**
  - **LEFT OUTER JOIN: Returns all rows from the left table and the matched rows from the right table. If no match is found, NULL values are returned for columns from the right table.**

    - **RIGHT OUTER JOIN: Returns all rows from the right table and the matched rows from the left table. If no match is found, NULL values are returned for columns from the left table.**
    - **FULL OUTER JOIN: Returns all rows when there is a match in either left or right table. Rows without a match in the other table will have NULL values for that table's columns.**
- **Examples: LEFT OUTER JOIN:**

  SQL

  .

  ```
  SELECT e.EmployeeID, e.FirstName, d.DepartmentName
  FROM Employees e
  LEFT OUTER JOIN Departments d ON e.DepartmentID = d.DepartmentID;
  ```

  This query retrieves all employees along with their department names. If an employee does not belong to any department, the DepartmentName will be NULL.

  RIGHT OUTER JOIN:

  SQL

  .

  ```
  SELECT e.EmployeeID, e.FirstName, d.DepartmentName
  FROM Employees e
  RIGHT OUTER JOIN Departments d ON e.DepartmentID = d.DepartmentID;
  ```

  This query retrieves all departments and their employees. If a department has no employees, the employee details will be NULL.

  FULL OUTER JOIN:

  SQL

  .

  ```
  SELECT e.EmployeeID, e.FirstName, d.DepartmentName
  FROM Employees e
  FULL OUTER JOIN Departments d ON e.DepartmentID = d.DepartmentID;
  ```

  This query retrieves all employees and all departments. If there’s no match between the tables, NULL values are returned for the missing data from either side.

2. OVER

- **Description: The OVER keyword is used in SQL to define a window or a set of rows for window functions (like ROW_NUMBER, RANK, SUM, AVG, etc.) to operate on. It allows you to perform calculations across a set of table rows that are somehow related to the current row.**
- **Examples:**

ROW_NUMBER:

SQL

.

```
SELECT EmployeeID, FirstName, Salary,
      ROW_NUMBER() OVER (ORDER BY Salary DESC) AS RowNum
FROM Employees;
```

This query assigns a unique row number to each employee, ordered by salary in descending order.

PARTITION BY:

SQL

.

```
SELECT DepartmentID, EmployeeID, Salary,
      RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS
SalaryRank
FROM Employees;
```

This query ranks employees within each department by salary. The PARTITION BY clause divides the result set into partitions, and the RANK function is applied within each partition.

AGGREGATE FUNCTION:

SQL

.

```
SELECT EmployeeID, FirstName, Salary,
      SUM(Salary) OVER (PARTITION BY DepartmentID ORDER BY Salary DESC)
AS RunningTotal
FROM Employees;
```

This query calculates a running total of salaries for each department, ordered by salary within the department.

These keywords, OUTER and OVER, are essential for complex data retrieval operations, especially when dealing with relationships between tables and performing calculations over specific data sets.

4o

P

- **PERCENT**
- **PIVOT**
- **PLAN**
- **PRECISION**
- **PRIMARY**
- **PRINT**
- **PROC**
- **PROCEDURE**
- **PUBLIC**

1. PERCENT

- **Description: The PERCENT keyword is used with the TOP clause to specify a percentage of rows to be returned from the result set. Instead of a fixed number of rows, you get a percentage of the total number of rows.**
- **Example:**

  SQL

  .

  ```
  SELECT TOP 10 PERCENT *
  FROM Employees
  ORDER BY Salary DESC;
  ```

  This query retrieves the top 10% of employees with the highest salaries.

2. PIVOT

- **Description: The PIVOT keyword is used to transform rows into columns in SQL, effectively rotating a table. It is useful for generating reports where data needs to be displayed in a cross-tab format.**
- **Example:**

  SQL

  .

```
SELECT * FROM
(SELECT Year, Quarter, SalesAmount FROM Sales) AS SourceTable
PIVOT
(
   SUM(SalesAmount)
   FOR Quarter IN ([Q1], [Q2], [Q3], [Q4])
) AS PivotTable;
```

  This query transforms the Sales data, rotating the Quarter values to become columns and showing the total sales for each quarter.

3. PLAN

- **Description: The PLAN keyword is used in certain SQL dialects (like Oracle) to refer to execution plans, which are descriptions of how SQL queries will be executed by the database engine. It's often seen in commands or tools related to performance tuning.**
- **Example:**

  SQL

  .

```
EXPLAIN PLAN FOR
SELECT * FROM Employees WHERE DepartmentID = 5;
```

  This command shows the execution plan for the query, detailing how the database will retrieve the data.

4. PRECISION

- **Description: The PRECISION keyword is used to define the number of digits in a numeric data type, such as DECIMAL or NUMERIC. It specifies the total number of digits that can be stored.**
- **Example:**

  SQL

  .

```
CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    Price DECIMAL(10, 2) -- 10 digits in total, 2 of which are after the decimal
point
);
```

This table defines the Price column with a precision of 10 and a scale of 2, meaning it can store values like 12345678.90.

5. PRIMARY

- **Description:** The PRIMARY keyword is used in conjunction with KEY to define a primary key constraint on a table. The primary key uniquely identifies each record in a table.
- **Example:**

```
SQL
.
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100)
);
```

This query creates a table with CustomerID as the primary key, ensuring that each customer has a unique identifier.

6. PRINT

- **Description:** The PRINT keyword is used to send a message to the SQL Server client. It's often used in stored procedures or scripts to provide status messages or debugging information.
- **Example:**

```
SQL
.
PRINT 'Operation completed successfully!';
```

This command prints the message to the output console of the SQL Server.

7. PROC

- **Description: The PROC keyword is an abbreviation for PROCEDURE and is used interchangeably with it. It's used to define or execute a stored procedure.**
- **Example:**

SQL

.

```
CREATE PROC GetEmployeeDetails
AS
BEGIN
   SELECT * FROM Employees;
END;
```

This command creates a stored procedure that retrieves all employee details.

8. PROCEDURE

- **Description: The PROCEDURE keyword is used to define a stored procedure, which is a set of SQL statements that can be executed as a unit. Stored procedures are used to encapsulate repetitive tasks or complex logic.**
- **Example:**

SQL

.

```
CREATE PROCEDURE UpdateEmployeeSalary
@EmployeeID INT,
@NewSalary DECIMAL(10, 2)
AS
BEGIN
   UPDATE Employees
   SET Salary = @NewSalary
   WHERE EmployeeID = @EmployeeID;
END;
```

This stored procedure updates the salary of an employee based on the provided EmployeeID and NewSalary.

9. PUBLIC

- **Description: The PUBLIC keyword is used in SQL to refer to a default role or to specify permissions that apply to all users. It's commonly used in the context of granting or revoking permissions.**
- **Example:**

  SQL

  .

  ```
  GRANT SELECT ON Employees TO PUBLIC;
  ```

  This command grants the SELECT permission on the Employees table to all users (i.e., the PUBLIC role).

R

- **RAISERROR**
- **READ**
- **READTEXT**
- **RECONFIGURE**
- **REFERENCES**
- **REPLICATION**
- **RESTORE**
- **RESTRICT**
- **RETURN**
- **REVERT**
- **REVOKE**
- **RIGHT**
- **ROLLBACK**
- **ROWCOUNT**
- **ROWGUIDCOL**
- **RULE**

1. RAISERROR

- **Description: The RAISERROR statement in SQL Server generates an error message that is returned to the application or client. It's often used in stored procedures or triggers to handle errors and send custom messages.**
- **Example:**

  SQL

  .

  ```
  RAISERROR ('Invalid operation: Division by zero.', 16, 1);
  ```

This raises an error with a custom message and a severity level of 16.

2. READ

- **Description: The READ keyword is used in SQL Server in the context of locking mechanisms, often combined with READONLY to indicate that a cursor or a table is read-only and cannot be modified.**
- **Example:**

  SQL
  .

  ```
  DECLARE cursor_example CURSOR READONLY
  FOR SELECT FirstName, LastName FROM Employees;
  ```

  This declares a read-only cursor, meaning the data retrieved by the cursor cannot be modified.

3. READTEXT

- **Description: The READTEXT statement is used in SQL Server to read text, ntext, or image data from a specified starting point in a column. It is mainly used for large text or binary data stored in these specific data types.**
- **Example:**

  SQL
  .

  ```
  READTEXT MyTable.MyColumn @ptrval 0 100;
  ```

  This reads 100 bytes of data from the MyColumn column in the MyTable table starting at the pointer value @ptrval.

4. RECONFIGURE

- **Description: The RECONFIGURE command is used in SQL Server to apply any changes made to the server's configuration options using sp_configure. It tells the server to update the configuration with the new settings.**
- **Example:**

  SQL
  .

```
EXEC sp_configure 'max server memory', 4096;
RECONFIGURE;
```

This sets the maximum server memory to 4096 MB and applies the change.

5. REFERENCES

- Description: The REFERENCES keyword is used to define a foreign key relationship between two tables. It specifies that a column (or set of columns) in one table must match the primary key (or unique key) of another table.
- Example:

  SQL

  .

  ```
  CREATE TABLE Orders (
      OrderID INT PRIMARY KEY,
      CustomerID INT,
      FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
  );
  ```

  This creates an Orders table where the CustomerID column references the CustomerID column in the Customers table.

6. REPLICATION

- Description: The REPLICATION keyword is used in the context of database replication, which is the process of copying and distributing data and database objects from one database to another and keeping them synchronized.
- Example:

  SQL

  .

  ```
  -- Example related to replication
  SELECT * FROM MSreplication_subscriptions;
  ```

  This would show subscriptions related to database replication.

7. RESTORE

- **Description: The RESTORE command is used to restore a database from a backup. It's essential for database recovery.**
- **Example:**

SQL

.

```
RESTORE DATABASE MyDatabase
FROM DISK = 'C:\Backup\MyDatabase.bak'
WITH RECOVERY;
```

This restores the MyDatabase from a backup file and brings it online with the WITH RECOVERY option.

8. RESTRICT

- **Description: The RESTRICT keyword is used in SQL as an option for DROP and ALTER commands to prevent the dropping or altering of an object if it has dependencies.**
- **Example:**

SQL

.

```
ALTER TABLE Orders
DROP COLUMN CustomerID RESTRICT;
```

This command would fail if there are dependencies on the CustomerID column, because of the RESTRICT option.

9. RETURN

- **Description: The RETURN keyword is used in stored procedures or functions to exit the procedure or function and optionally return a value to the calling program.**
- **Example:**

SQL

.

```
CREATE PROCEDURE CheckEmployee
@EmployeeID INT
AS
BEGIN
    IF @EmployeeID IS NULL
    BEGIN
```

```
        RETURN -1; -- Indicating an error
    END
    RETURN 0; -- Indicating success
END;
```

This stored procedure checks if an EmployeeID is null and returns -1 if it is, or 0 otherwise.

10. REVERT

- **Description: The REVERT statement is used in SQL Server to return to the previous security context after an EXECUTE AS statement has been used to switch to another user context.**
- **Example:**

```
SQL
.
EXECUTE AS USER = 'JohnDoe';
-- Do some operations as JohnDoe
REVERT;
```

This switches the security context to JohnDoe and then reverts to the original context.

11. REVOKE

- **Description: The REVOKE command is used to remove previously granted permissions from a user or role. It's a way to ensure that a user or role no longer has specific privileges.**
- **Example:**

```
SQL
.
REVOKE SELECT ON Employees FROM JohnDoe;
```

This revokes the SELECT permission on the Employees table from the user JohnDoe.

12. RIGHT

- **Description: The RIGHT keyword is used in SQL to perform a RIGHT JOIN between two tables. This type of join returns all rows from the right**

table and the matching rows from the left table. If there’s no match, the result will include NULL values for the left table.

- **Example:**

SQL

.

```
SELECT e.FirstName, d.DepartmentName
FROM Employees e
RIGHT JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

This query returns all departments and their employees. If a department has no employees, the employee details will be NULL.

13. ROLLBACK

- **Description: The ROLLBACK statement is used to undo a transaction that has not yet been committed, restoring the database to its previous state.**
- **Example:**

SQL

.

```
BEGIN TRANSACTION;
DELETE FROM Employees WHERE EmployeeID = 5;
ROLLBACK;
```

This starts a transaction, deletes an employee record, and then rolls back the transaction, undoing the delete.

14. ROWCOUNT

- **Description: The ROWCOUNT keyword in SQL Server limits the number of rows affected by a query, such as INSERT, UPDATE, or DELETE. It can also be used to retrieve the number of rows affected by the last operation.**
- **Example:**

SQL

.

```
SET ROWCOUNT 5;
DELETE FROM Employees WHERE DepartmentID = 2;
SET ROWCOUNT 0;
```

This deletes only the first 5 rows where DepartmentID is 2. The ROWCOUNT is then reset to its default behavior.

15. ROWGUIDCOL

- Description: The ROWGUIDCOL keyword is used to mark a uniqueidentifier column in a table as the row globally unique identifier (GUID) column. This column is typically used to uniquely identify rows in a database.
- Example:

```
SQL
.
CREATE TABLE Products (
   ProductID UNIQUEIDENTIFIER ROWGUIDCOL DEFAULT NEWID(),
   ProductName VARCHAR(50)
);
```

This creates a table where the ProductID is automatically assigned a unique GUID value.

16. RULE

- Description: The RULE keyword is used in SQL Server to create or apply a rule that specifies valid values for a column or a user-defined data type. However, RULE is deprecated in favor of CHECK constraints.
- Example:

```
SQL
.
CREATE RULE PositiveSalary AS @Salary > 0;
```

This rule enforces that the salary must be greater than 0.

S

❖

❖ SAVE

❖ SCHEMA

❖ SECURITYAUDIT

❖ SELECT

- **SEMANTICKEYPHRASETABLE**
- **SEMANTICSIMILARITYDETAILSTABLE**
- **SEMANTICSIMILARITYTABLE**
- **SESSION_USER**
- **SET**
- **SETUSER**
- **SHUTDOWN**
- **SOME**
- **STATISTICS**
- **SYSTEM_USER**

1. SAVE

- **Description: The SAVE keyword is used in SQL Server to create a savepoint within a transaction. A savepoint allows you to roll back part of a transaction without rolling back the entire transaction.**
- **Example:**

SQL

.

```
BEGIN TRANSACTION;
DELETE FROM Orders WHERE OrderID = 1;
SAVE TRANSACTION SavePoint1;
DELETE FROM Orders WHERE OrderID = 2;
ROLLBACK TRANSACTION SavePoint1;
COMMIT TRANSACTION;
```

In this example, the deletion of OrderID = 2 is rolled back, but the deletion of OrderID = 1 is committed.

2. SCHEMA

- **Description: The SCHEMA keyword is used to define a schema, which is a logical collection of database objects like tables, views, and stored procedures. Schemas help organize and manage these objects within a database.**
- **Example:**

SQL

.

```
CREATE SCHEMA Sales;
CREATE TABLE Sales.Customers (
   CustomerID INT PRIMARY KEY,
   CustomerName VARCHAR(100)
);
```

This creates a Sales schema and a Customers table within that schema.

3. SECURITYAUDIT

- Description: The SECURITYAUDIT keyword is related to SQL Server's security auditing, which is used to track and log security-related actions. It's a reserved keyword, but it does not have a specific command or usage in standard SQL queries.
- Example: While SECURITYAUDIT is reserved for future use and does not have direct usage, SQL Server provides auditing through other means, such as configuring SQL Server Audit.

4. SELECT

- Description: The SELECT statement is used to retrieve data from one or more tables in a database. It's one of the most fundamental and commonly used SQL commands.
- Example:

SQL

.

```
SELECT FirstName, LastName FROM Employees WHERE DepartmentID = 5;
```

This query selects the FirstName and LastName columns from the Employees table where the DepartmentID is 5.

5. SEMANTICKEYPHRASETABLE

- Description: The SEMANTICKEYPHRASETABLE function in SQL Server is used to extract key phrases from a text column. It's part of the full-text search and semantic search features.
- Example:

SQL

.

```
SELECT * FROM SEMANTICKEYPHRASETABLE(Products, ProductDescription);
```

This would extract key phrases from the ProductDescription column in the Products table.

6. SEMANTICSIMILARITYDETAILSTABLE

- **Description: The SEMANTICSIMILARITYDETAILSTABLE function returns a table that provides details about how similar two rows of text data are. It's used for deep analysis of text similarity.**
- **Example:**

SQL

.

```
SELECT * FROM SEMANTICSIMILARITYDETAILSTABLE(Products, ProductID,
ProductDescription);
```

This would return details on the similarity between rows based on the ProductDescription column.

7. SEMANTICSIMILARITYTABLE

- **Description: The SEMANTICSIMILARITYTABLE function returns a table of rows that are semantically similar to the specified row in a text column.**
- **Example:**

SQL

.

```
SELECT * FROM SEMANTICSIMILARITYTABLE(Products, ProductID,
ProductDescription);
```

This query would return rows that are semantically similar to the text in the ProductDescription column.

8. SESSION_USER

- **Description: The SESSION_USER function returns the name of the current user for the session. It's similar to CURRENT_USER and is often used to track or audit user actions.**
- **Example:**

SQL

.

```
SELECT SESSION_USER AS CurrentSessionUser;
```

This would return the name of the current session user.

9. SET

- **Description: The SET statement is used to assign a value to a variable or to set an option for the session, such as transaction isolation level or the language.**
- **Example:**

```
SQL
.
DECLARE @TotalSales INT;
SET @TotalSales = (SELECT SUM(SalesAmount) FROM Sales);
```

This sets the value of the @TotalSales variable to the sum of the SalesAmount from the Sales table.

10. SETUSER

- **Description: The SETUSER command was used in SQL Server to impersonate another user. However, it has been deprecated in favor of EXECUTE AS.**
- **Example:**

```
SQL
.
EXECUTE AS USER = 'JohnDoe';
-- Perform actions as JohnDoe
REVERT;
```

This is the modern way to switch user context and then revert back to the original user.

11. SHUTDOWN

- **Description: The SHUTDOWN command is used to shut down the SQL Server instance. This is usually restricted to administrative users.**
- **Example:**

```
SQL
.
SHUTDOWN;
```

This would stop the SQL Server service.

12. SOME

- **Description: The SOME keyword is used with comparison operators to return true if at least one of the subquery values meets the condition. It's similar to ANY.**
- **Example:**

  SQL

  .

  ```
  SELECT * FROM Employees WHERE Salary > SOME (SELECT Salary FROM Managers);
  ```

  This query selects employees whose salary is greater than at least one of the manager's salaries.

13. STATISTICS

- **Description: The STATISTICS keyword is used in SQL Server to refer to the automatic or manual collection of data distribution statistics, which help the query optimizer make better decisions.**
- **Example:**

  SQL

  .

  ```
  UPDATE STATISTICS Employees;
  ```

  This command updates the statistics for the Employees table, helping the optimizer with future queries.

14. SYSTEM_USER

- **Description: The SYSTEM_USER function returns the login name of the user currently connected to SQL Server. It's often used for auditing and security purposes.**
- **Example:**

  SQL

  .

  ```
  SELECT SYSTEM_USER AS LoggedInUser;
  ```

  This would return the name of the currently logged-in user.

T

- ❖ TABLE
- ❖ TABLESAMPLE
- ❖ TEXTSIZE
- ❖ THEN
- ❖ TO

1. TABLE

- Description: The TABLE keyword is fundamental in SQL, used to create, modify, or reference a table in a database. Tables are collections of related data organized into rows and columns.
- Example:

  SQL

  .

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    DepartmentID INT
);
```

  This creates a new table named Employees with columns for EmployeeID, FirstName, LastName, and DepartmentID.

2. TABLESAMPLE

- Description: The TABLESAMPLE clause is used to retrieve a random sample of rows from a table, which can be useful for testing or working with large datasets where you don't need the full data.
- Example:

  SQL

  .

```
SELECT * FROM Employees TABLESAMPLE (10 PERCENT);
```

  This query returns approximately 10% of the rows from the Employees table randomly.

3. TEXTSIZE

- **Description: The TEXTSIZE option in SQL Server specifies the maximum size of text, ntext, or image data that can be returned with a SELECT statement. It controls the amount of text data retrieved.**
- **Example:**

SQL

.

```
SET TEXTSIZE 500;
SELECT Notes FROM EmployeeNotes WHERE EmployeeID = 1;
```

This sets the TEXTSIZE to 500 characters, meaning that only the first 500 characters of the Notes column will be returned in the query.

4. THEN

- **Description: The THEN keyword is used in conditional statements, such as CASE, to define the action or value that should be returned when a condition is met.**
- **Example:**

SQL

.

```
SELECT
   FirstName,
   LastName,
   CASE
      WHEN DepartmentID = 1 THEN 'HR'
      WHEN DepartmentID = 2 THEN 'Sales'
      ELSE 'Other'
   END AS DepartmentName
FROM Employees;
```

This query uses THEN in a CASE statement to return a department name based on the DepartmentID.

5. TO

- **Description: The TO keyword is used in various SQL commands to indicate a target or destination, such as in GRANT, REVOKE, ALTER, and data type conversions.**
- **Example:**

SQL

.

```
GRANT SELECT ON Employees TO JohnDoe;
```

This grants SELECT permissions on the Employees table to the user JohnDoe.

- TOP
- TRAN
- TRANSACTION
- TRIGGER
- TRUNCATE
- TRY_CONVERT
- TSEQUAL

1. TOP

- Description: The TOP keyword is used to limit the number of rows returned in a result set. It is often used with the SELECT statement to retrieve a specific number of records.
- Example:

SQL

.

```
SELECT TOP 5 * FROM Employees ORDER BY HireDate DESC;
```

This query retrieves the top 5 most recently hired employees from the Employees table.

2. TRAN

- Description: TRAN is a shorthand for TRANSACTION. It is used to start a transaction block in SQL Server. Transactions allow you to group a set of operations that can be committed or rolled back as a single unit.
- Example:

SQL

.

```
BEGIN TRAN;
DELETE FROM Orders WHERE OrderID = 10;
-- If something goes wrong
```

```
ROLLBACK TRAN;
-- Otherwise
COMMIT TRAN;
```

This starts a transaction, deletes an order, and then either commits or rolls back the transaction based on further conditions.

3. TRANSACTION

- Description: The TRANSACTION keyword is used to define a transaction block. Transactions ensure that a series of SQL statements are executed as a single unit, providing a way to ensure data integrity.
- Example:

```
SQL
.
BEGIN TRANSACTION;
UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;
UPDATE Accounts SET Balance = Balance + 100 WHERE AccountID = 2;
COMMIT TRANSACTION;
```

This transfers $100 from one account to another within a transaction, ensuring that both updates either succeed or fail together.

4. TRIGGER

- Description: A TRIGGER is a special type of stored procedure that automatically executes when a specific event occurs in the database, such as an INSERT, UPDATE, or DELETE.
- Example:

```
SQL
.
CREATE TRIGGER trgAfterInsert ON Employees
AFTER INSERT
AS
BEGIN
   INSERT INTO EmployeeAudit(EmployeeID, ChangeDate, Action)
   SELECT EmployeeID, GETDATE(), 'INSERT'
   FROM inserted;
END;
```

This trigger automatically logs an insert action into an EmployeeAudit table whenever a new row is added to the Employees table.

5. TRUNCATE

- **Description:** The TRUNCATE statement is used to remove all rows from a table without logging the individual row deletions, making it faster than DELETE for large tables. However, TRUNCATE cannot be used with tables that have foreign key constraints.
- **Example:**

SQL

.

```
TRUNCATE TABLE Employees;
```

This command removes all records from the Employees table, but the structure of the table remains.

6. TRY_CONVERT

- **Description:** The TRY_CONVERT function attempts to convert an expression from one data type to another. If the conversion fails, it returns NULL instead of throwing an error, making it safer for certain operations.
- **Example:**

SQL

.

```
SELECT TRY_CONVERT(INT, '1234') AS Result1,
       TRY_CONVERT(INT, 'ABC') AS Result2;
```

In this example, Result1 would return 1234, while Result2 would return NULL because the string 'ABC' cannot be converted to an integer.

7. TSEQUAL

- **Description:** The TSEQUAL function is a legacy SQL Server function used to compare two timestamp or rowversion values to determine if they are equal. It is rarely used in modern SQL.
- **Example:**

SQL

```
.
IF TSEQUAL(@OldTimestamp, @NewTimestamp)
   PRINT 'Timestamps are equal';
ELSE
   PRINT 'Timestamps are not equal';
```

This example compares two timestamp values and prints a message indicating whether they are equal.

U

- **UNION**
- **UNIQUE**
- **UNPIVOT**
- **UPDATE**
- **UPDATETEXT**

- **USE**
- **USER**

1. UNION

- **Description: The UNION operator is used to combine the results of two or more SELECT statements into a single result set. It removes duplicate rows by default.**
- **Example:**

SQL

```
.
SELECT FirstName, LastName FROM Employees
UNION
SELECT FirstName, LastName FROM Managers;
```

This query combines the results of Employees and Managers tables, returning a list of unique first and last names from both.

2. UNIQUE

- **Description: The UNIQUE constraint ensures that all values in a column or a set of columns are distinct, meaning no duplicate values are allowed.**

- **Example:**

  SQL

  .

  ```
  CREATE TABLE Customers (
      CustomerID INT PRIMARY KEY,
      Email VARCHAR(100) UNIQUE
  );
  ```

  This creates a Customers table where the Email column must have unique values—no two customers can have the same email address.

3. UNPIVOT

- **Description: The UNPIVOT operator transforms columns into rows, effectively the reverse of PIVOT. It is used to normalize data that has been denormalized.**
- **Example:**

  SQL

  .

  ```
  SELECT EmployeeID, Attribute, Value
  FROM
  (
      SELECT EmployeeID, Q1Sales, Q2Sales, Q3Sales, Q4Sales
      FROM QuarterlySales
  ) AS SalesData
  UNPIVOT
  (
      Value FOR Attribute IN (Q1Sales, Q2Sales, Q3Sales, Q4Sales)
  ) AS UnpivotedData;
  ```

  This query transforms quarterly sales data from columns (Q1Sales, Q2Sales, etc.) into rows.

4. UPDATE

- **Description: The UPDATE statement is used to modify existing records in a table. You can update one or more columns for all rows or specific rows that meet a condition.**
- **Example:**

  SQL

.

```
UPDATE Employees
SET Salary = Salary * 1.05
WHERE DepartmentID = 3;
```

This query increases the salary by 5% for all employees in department 3.

5. UPDATETEXT

- **Description:** The UPDATETEXT statement is used to update a part of a text, ntext, or image data type in SQL Server. It allows for in-place updates of large text data, but it is deprecated in favor of other techniques.
- **Example:**

  SQL

  .

  ```
  DECLARE @ptrval BINARY(16);
  SELECT @ptrval = TEXTPTR(Notes) FROM EmployeeNotes WHERE EmployeeID
  = 1;
  UPDATETEXT EmployeeNotes.Notes @ptrval 0 0 'New notes: ';
  ```

  This updates the Notes column for the specified employee by inserting the string 'New notes: ' at the beginning.

6. USE

- **Description:** The USE statement is used to switch the context of the current session to a specified database. It allows you to run SQL commands in the context of a different database.
- **Example:**

  SQL

  .

  ```
  USE SalesDB;
  SELECT * FROM Orders;
  ```

  This command switches to the SalesDB database, and then selects all records from the Orders table.

7. USER

- **Description: The USER function returns the current database user name. It is used to identify the user executing a query or script.**
- **Example:**

SQL

.

```
SELECT USER AS CurrentUser;
```

This query returns the name of the current database user.

V

- **VALUES**
- **VARYING**
- **VIEW**

1. VALUES

- **Description: The VALUES keyword is used in INSERT statements to specify the data that will be inserted into a table. It allows you to insert one or more rows of data at once.**
- **Example:**

SQL

.

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID)
VALUES (1, 'John', 'Doe', 3),
       (2, 'Jane', 'Smith', 2);
```

This INSERT statement adds two new records into the Employees table with specified values for EmployeeID, FirstName, LastName, and DepartmentID.

2. VARYING

- **Description: The VARYING keyword is often used in conjunction with data types like VARCHAR (variable character) or VARBINARY (variable binary) to define columns that can store strings or binary data of variable length up to a specified limit.**
- **Example:**

SQL

.

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100) VARYING
);
```

In this example, VARCHAR(100) indicates that the CustomerName column can store up to 100 characters of variable length.

3. VIEW

- **Description:** A VIEW is a virtual table based on the result set of a SELECT query. It does not store the data physically but allows you to simplify complex queries, secure data by limiting access to specific columns, and present data in a specific format.
- **Example:**

SQL

.

```
CREATE VIEW EmployeeDetails AS
SELECT
    e.EmployeeID,
    e.FirstName,
    e.LastName,
    d.DepartmentName
FROM Employees e
JOIN Departments d ON e.DepartmentID = d.DepartmentID;
```

This creates a view named EmployeeDetails that combines information from the Employees and Departments tables, showing employee names along with their department names.

W

- WAITFOR
- WHEN
- WHERE
- WHILE
- WITH
- WRITETEXTSQL

1. WAITFOR

- **Description: The WAITFOR keyword is used to delay the execution of a query or stored procedure for a specified time or until a specific event occurs.**
- **Example:**

SQL

.

```
WAITFOR DELAY '00:00:10'; -- Waits for 10 seconds
```

This command pauses the execution for 10 seconds.

SQL

.

```
WAITFOR TIME '23:59:59'; -- Waits until a specific time
```

This command pauses execution until 11:59:59 PM.

SQL

.

```
WAITFOR (RECEIVE * FROM MyQueue); -- Waits for a message to arrive in a queue
```

This command waits for a message to arrive in the MyQueue service broker queue.

2. WHEN

- **Description: The WHEN keyword is used in CASE statements to define the condition that determines which result to return.**
- **Example:**

SQL

.

```
SELECT FirstName, LastName,
       CASE
          WHEN Salary > 50000 THEN 'High'
          WHEN Salary BETWEEN 30000 AND 50000 THEN 'Medium'
          ELSE 'Low'
       END AS SalaryCategory
```

```
FROM Employees;
```

This query categorizes employees based on their salary using WHEN to define conditions.

3. WHERE

- Description: The WHERE keyword is used to filter records based on specified conditions. It limits the rows returned by a query.
- Example:

```
SQL
.
SELECT * FROM Employees
WHERE Department = 'Sales';
```

This query retrieves all employees who are in the Sales department.

```
SQL
.
SELECT * FROM Orders
WHERE OrderDate >= '2023-01-01';
```

This query retrieves orders that have been placed on or after January 1, 2023.

4. WHILE

- Description: The WHILE keyword is used to execute a block of statements repeatedly while a specified condition is TRUE.
- Example:

```
SQL
.
DECLARE @Counter INT = 0;
WHILE @Counter < 5
BEGIN
   PRINT 'Counter value: ' + CAST(@Counter AS VARCHAR);
   SET @Counter = @Counter + 1;
END;
```

This script prints the value of @Counter and increments it until @Counter reaches 5.

5. WITH

- **Description:** The WITH keyword is used to define Common Table Expressions (CTEs), temporary result sets that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. It is also used to specify table hints in some SQL dialects.
- **Example:**

SQL

.

```
WITH EmployeeCTE AS (
   SELECT EmployeeID, FirstName, LastName
   FROM Employees
   WHERE Department = 'IT'
)
SELECT * FROM EmployeeCTE;
```

This query defines a CTE EmployeeCTE to select employees from the IT department and then retrieves all rows from that CTE.

SQL

.

```
SELECT * FROM Orders WITH (NOLOCK);
```

This query uses the WITH clause to specify a table hint NOLOCK, which allows reading data without acquiring shared locks.

6. WRITETEXT

- **Description:** The WRITETEXT statement is used to modify text or image data in SQL Server. It allows appending or updating the data in a TEXT, NTEXT, or IMAGE column.
- **Example:**

SQL

.

```
WRITETEXT Documents.DocumentText 'New text data';
```

This command appends the specified text data to the DocumentText column in the Documents table.

SQL

.

```
UPDATETEXT Documents.DocumentText NULL NULL 'Additional text data';
```

This command updates the DocumentText column with additional text data.

Note: TEXT, NTEXT, and IMAGE data types are deprecated in SQL Server; consider using VARCHAR(MAX), NVARCHAR(MAX), or VARBINARY(MAX) instead.

X

- XOR

XOR

- Description: XOR (exclusive OR) is a logical operator used to return TRUE if exactly one of the conditions is TRUE, and FALSE otherwise. It is often used in conditional statements or expressions to evaluate whether exactly one of two conditions is met.
- Usage in SQL: SQL doesn’t have a direct XOR operator, but you can achieve similar results using logical conditions. In most SQL dialects, you can use a combination of AND, OR, and NOT to simulate XOR.
- Examples:

1. Simulating XOR in SQL:

SQL

.

```
SELECT *
FROM TableName
WHERE (Condition1 = TRUE AND Condition2 = FALSE)
   OR (Condition1 = FALSE AND Condition2 = TRUE);
```

In SQL, while there isn't a direct XOR operator, you can use logical expressions to achieve similar outcomes. Let me know if you need further explanations or examples!

Y

- **YEAR**

YEAR

- **Description: The `YEAR` function is used to extract the year part from a date or datetime value in SQL. It is commonly used in queries to filter or group data based on the year.**
- **Example:**

  1. Extracting the Year from a Date:

  SQL

  .

  ```
  SELECT YEAR(OrderDate) AS OrderYear
  FROM Orders;
  ```

  This query retrieves the year portion of the `OrderDate` column for each row in the `Orders` table, and labels it as `OrderYear`.

  2. Filtering Data by Year:

  SQL

  .

  ```
  SELECT *
  FROM Orders
  WHERE YEAR(OrderDate) = 2023;
  ```

  This query returns all orders where the `OrderDate` falls in the year 2023.

  3. Grouping Data by Year:

  SQL

  .

  ```
  SELECT YEAR(OrderDate) AS OrderYear, COUNT(*) AS NumberOfOrders
  FROM Orders
  GROUP BY YEAR(OrderDate);
  ```

This query groups the orders by year and counts the number of orders for each year.

4. Using YEAR in a Calculation:

SQL

.

```
SELECT ProductName, YEAR(ManufactureDate) AS ManufactureYear
FROM Products
WHERE YEAR(ManufactureDate) > 2020;
```

This query selects product names and their manufacture years for products manufactured after the year 2020.