SQL Structured Query Language (Northwind DB)

Fatemeh Heydari

SQL is a powerful language used for managing and manipulating relational databases, enabling efficient data storage, retrieval, and analysis.

Select Syntax:

Select <Exp. List>
From <Table Name> | <View Name> | <Derived Table> | <CTE> | <Table Valued Function> | <Synonym>
[**Where** <Boolean Exp.>]
[**Group By** <Exp. List>]
[**Having** <Boolean Exp.>]
[**Order By** <Exp. List> [ASC| DESC]]

EX: Retrieve all columns and rows from the Products table.

```
Select *
From Products
```

EX: Retrieve Product name from the Products table where the unit price is greater than 10.

```
Select ProductName
From Products Where UnitPrice>10
```

EX: Retrieve all columns from the Products table, ordering the rows by ascending unit price and descending UnitsInStock for items with the same unit price.

```
SELECT *
From Products
Order BY UnitPrice ASC,
UnitsInStock DESC
```

Select Cluse:

SELECT [ALL | DISTINCT]
[TOP (exp) [PERCENT] [WITH TIES]]

www.linkedin.com/in/fatemeh-heydari-49a2b3238

- **All:** Specifies that duplicate rows can appear in the result set. ALL is the default.

  EX:

  ```
  Select Country
  From Customers
  ```

  Result = 91 Rows

- **Distinct:** Specifies that only unique rows can appear in the result set.

  EX:

  ```
  Select Distinct Country
  From Customers
  ```

  Result = 21 Unique Rows

- **Top:** limit the number or percentage of rows returned from the query result set.

  EX:

  ```
  SELECT TOP 50 PERCENT *
  FROM Customers
  ```

  Result = 46 Rows (50% of the records from the "Customers" table)

  EX:

  ```
  Select TOP 12 With Ties *
  From Products
  Order By UnitPrice ASC
  ```

  Result = 14 Rows (Because records 12, 13 and 14 have the same Unit Price)

Alias:

An alias in SQL is an alternative name used for referencing tables, columns, or expressions, improving query readability.

- **Columns:**

  ```
  Select ProductName As "Product Name"
  From Products
  ```

- **Tables:**

  ```
  Select ProductName
  From Products As P
  ```

- **Expression:**

  ```
  Select UnitPrice*Quantity As
  GrossAmount
  From [Order Details]
  ```

Expression in a SELECT statement Can be:

- **Variable**
- **Column**
- **Scalar function**
- ➢ Operators can be used to join two or more simple expressions into a complex expression.

Scalar Functions:

- **Mathematical Functions:**
  - **Floor** (numeric_exp)

    Returns the largest integer less than or equal to the specified numeric expression.

    ```
    Select Floor (3.89)
    ```

    Result = 3

  - **Celling** (numeric_exp)

    Returns the smallest integer greater than, or equal to, the specified numeric expression.

    ```
    Select CEILING (3.14)
    ```

    Result = 4

- **Conversion Functions:**

  Convert an expression of one data type to another.

  - **Cast** (Exp AS data_type [(length)])

    ```
    Select CAST (123 AS nvarchar (10))
    +'ABC'
    ```

    Result = 123ABC

- **Convert** (data_type [(length)], exp [, style])

```
Select CONVERT (nvarchar (5), 123)
+'ABC'
```

Result = 123ABC

- **String Functions:**
  - **Concat** (string_value1, string_value2 [, string_valueN])

```
Select CONCAT ('A', 'B', Null, 'C')
```

Result = ABC

  - **Right** (character_exp, integer_exp)

```
Select Right ('SQL Server', 6)
```

Result = Server

  - **Left** (character_exp, integer_exp)

```
Select Left ('SQL Server', 3)
```

Result = SQL

  - **Substring** (exp, start, length)

```
Select Substring ('SQL Server',1,3)
```

Result = SQL

  - **Reverse** (string_exp)

```
Select REVERSE('ABC')
```

Result = CBA

- **Date and time Functions:**
  - **DateDiff** (<Interval>, Start Date, End Date)

```
Select DATEDIFF (Day, '07-10-2023',
'07-12-2023')
```

Result = 2

- **Logical Functions:**
  - **IIF** (boolean_expression, true_value, false_value)

```
Select IIF (10<20, 'T', 'F')
```

Result = T

Arithmetic Operators:

Arithmetic operators run mathematical operations on two expressions of one or more data types.

- **Add (+):** Supports Numeric Data, String, and Datetime types.
- **Subtract (-):** Supports Numeric Data and Datetime types.
- **Multiply (*):** Supports Numeric Data types.
- **Divide (/):** Supports Numeric Data types.

Case Syntax:

CASE
WHEN when_exp **THEN** result_exp [...n]
[**ELSE** else_result_exp]
END

```
Select productID, ProductName, UnitPrice,
 Case
  When Unitprice<10 Then 'Low'
  When UnitPrice Between 10 And 50 Then 'Medium'
  When UnitPrice>50 Then 'High'
  Else 'Other'
 End
From Products
```

View:

Generates a virtual table based on a query, defining its columns and rows.

CREATE [OR ALTER] VIEW V_name
AS
SELECT <column1, column2, ...>
FROM <Table Name>
[**Where** <Condition.>]

```
Create View V_GremanCustomers
AS
  Select CustomerID, CompanyName, Country,
  City
  From Customers
  Where Country = 'Germany'
```

Derived Column:

Create new column values by applying expressions to transformation input columns.

```
Select FirstName+' '+LastName As FullName
From Employees
```

Derived Table:

A Derived Table is an inner query defined in the FROM clause of an outer query, without creating a separate database object.

```
Select *
From (Select CustomerID, CompanyName, Country,
City
          From Customers
          Where Country = 'Germany'
      ) AS GermanCustomers
```

CTE:

CTE is derived from a simple query and defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE or MERGE statement.

With CTE_Name
AS (
SELECT <column1, column2, ...>
FROM <Table Name>
[**Where** <Condition.>]
)
SELECT <column1, column2, ...>
From CTE_Name
[**Where** <Condition.>]

SQL Structured Query Language (Northwind DB)

Fatemeh Heydari

```
With GermanCustomers
AS(
    Select CustomerID, CompanyName, Country,
    City
    From Customers
    Where Country ='Germany'
  )
Select *
From GermanCustomers
```

Where Cluse:

[**Where** <Boolean Exp.>]

<Exp 1> <comparison Operator> <Exp2>

Comparison Operator:

=	Equals
>	Greater Than
>=	Greater Than or Equal To
<	Less Than
<=	Less Than or Equal To
<>	Not Equal To
!=	Not Equal To

Logical Operator:

- **AND:** Combines two Boolean expressions and returns TRUE when both expressions are TRUE.

```
Select *
From Products
Where UnitPrice>=10 AND UnitPrice<=20
```

- **OR:** Combines two conditions. When more than one logical operator is used in a statement, OR operators are evaluated after AND operators.

```
Select *
From Products
Where UnitPrice<10 OR UnitPrice>20
```

www.linkedin.com/in/fatemeh-heydari-49a2b3238

- **NOT:** Negates a Boolean input.

```
Select *
From Products
Where NOT (UnitPrice>=10 AND UnitPrice<=20)
```

- **BETWEEN:** Specifies a range to test.

```
Select *
From Products
Where UnitPrice BETWEEN 10 AND 20
```

- **IN:** Determines whether a specified value matches any value in a subquery or a list.

```
Select *
From Products
Where CategoryID IN (1,3,7)
```

- **LIKE:** Determines whether a specific character string matches a specified pattern.
  EX: retrieves all columns from the "Products" table where the ProductName ends with the letter 'S'.

```
Select *
From Products
Where ProductName LIKE '%S'
```

IS NULL:

Determines whether a specified expression is NULL.

Expression IS [NOT] NULL

```
Select *
From Customers
Where Region IS NOT NULL
```

Aggregate Functions:

- **Count** (<Exp>)

Return the number of items found in a group.

- ❖ **Count (*):** Return the number of total rows, regardless of the presence of NULL values.

```
Select Count (CustomerID)
From Customers
Where Country='UK'
```

- ***SUM*** (<Numeric Exp >)

Return the sum of all the values in the expression.

```
SELECT SUM(UnitsInStock) AS TotalStock
FROM Products
```

- ***AVG*** (<Numeric Exp >)

```
SELECT AVG(1.0 *UnitPrice) AS AveragePrice
FROM Products
```

```
SELECT AVG(CONVERT(decimal(10,2),
UnitPrice)) AS AveragePrice
FROM Products
```

- ***MIN*** (<Exp>)

```
SELECT MIN (UnitPrice)
FROM Products
```

- ***MAX*** (<Exp>)

```
SELECT MAX (UnitPrice)
FROM Products
```

- ***String-AGG*** (<String Exp>, <Separator>)

```
SELECT CategoryID, STRING_AGG(ProductName,
', ') AS ProductList
FROM Products
GROUP BY CategoryID
```

Count Distinct:

Return the number of unique nonnull values.

```
SELECT COUNT(DISTINCT CustomerID) AS
TotalDistinctCustomers
FROM Orders
```

SQL Structured Query Language (Northwind DB)

Fatemeh Heydari

Group By:

The GROUP BY statement combines rows with identical values into summary rows.

Syntax:

SELECT column1, column2, ..., Aggregate_Function (column)

FROM table_name

[**WHERE** condition]

GROUP BY column1, column2, ...

```
SELECT Country, Count (CustomerID) AS
CustomerCount
FROM Customers
GROUP BY Country
```

Having:

HAVING is used to filter query results based on conditions applied to aggregated data.

```
SELECT OrderID, SUM(Quantity*UnitPrice) AS
SalesAmount
FROM [Order Details]
GROUP BY OrderID
HAVING SUM(Quantity*UnitPrice) > 15000
ORDER BY OrderID
```

Use Case in Group By:

```
SELECT
CASE
       WHEN UnitsInStock < 10 THEN 'Low
       Stock'
       WHEN UnitsInStock BETWEEN 10 AND 50
       THEN 'Medium Stock'
       ELSE 'HighStock'
END AS StockCategory,
    COUNT(ProductID) AS ProductCount
FROM Products
GROUP BY
CASE
       WHEN UnitsInStock < 10 THEN 'Low
       Stock'
       WHEN UnitsInStock BETWEEN 10 AND 50
       THEN 'Medium Stock'
       ELSE 'HighStock'
END
```

www.linkedin.com/in/fatemeh-heydari-49a2b3238

Use Case in Aggregate Function:

```
SELECT EmployeeID,
    COUNT(OrderID) AS TotalOrders,
 CASE
    WHEN COUNT(OrderID) < 50 THEN 'Low Volume'
    WHEN COUNT(OrderID) BETWEEN 50 AND 100 THEN
    'Medium Volume'
    ELSE 'High Volume'
 END AS OrderVolumeClass
FROM Orders
GROUP BY EmployeeID
```

Group By Grouping Sets:

The GROUPING SETS allows you to consolidate multiple GROUP BY clauses into a single GROUP BY clause.

```
SELECT Country, City,
       COUNT(CustomerID) AS TotalCustomers
FROM
    Customers
GROUP BY
    GROUPING SETS (
       (Country, City),
          ()
    )
```

Querying Two Tables:

- **[INNER] JOIN:**

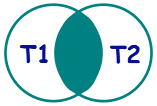
The INNER JOIN is used to select records that have matching values in both tables.

Syntax:

SELECT column1, column2, ...

FROM Table 1

INNER JOIN Table2 **ON** Table1.Pk = Table2.FK



Orders	
OrderID	CustomerID
10248	1
10249	2
10250	4

Customers	
CustomerID	CompanyName
1	A
2	B
3	C

```
SELECT C.CompanyName, O.OrderID
FROM Customers C INNER JOIN Orders O ON
C.CustomerID = O.CustomerID
```

Result:

CompanyName	OrderID
A	10248
B	10249

- **LEFT [OUTER] JOIN:**

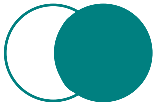
A LEFT OUTER JOIN retrieves records from the left table and their matching records from the right table, including NULL values from the right table if there is no match.

Syntax:

SELECT column1, column2, ...

FROM Table 1

LEFT JOIN Table2 **ON** Table1.Pk = Table2.FK



Orders	
OrderID	CustomerID
10248	1
10249	2
10250	4

Customers	
CustomerID	CompanyName
1	A
2	B
3	C

```
SELECT C.CompanyName, O.OrderID
FROM Customers C LEFT JOIN Orders O ON
C.CustomerID = O.CustomerID
```

Result:

CompanyName	OrderID
A	10248
B	10249
C	Null

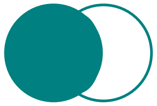
SQL Structured Query Language (Northwind DB)

Fatemeh Heydari

• RIGHT [OUTER] JOIN:

A RIGHT OUTER JOIN retrieves records from the right table and matching records from the left table, including NULL values for the columns from the left table if there is no match.

Syntax:
SELECT column1, column2, ...
FROM Table 1
RIGHT JOIN Table2 **ON** Table1.Pk = Table2.FK



Orders	
OrderID	CustomerID
10248	1
10249	2
10250	4

Customers	
CustomerID	CompanyName
1	A
2	B
3	C

```
SELECT C.CompanyName, O.OrderID
FROM Customers C RIGHT JOIN Orders O ON
C.CustomerID = O.CustomerID
```

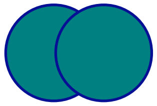
Result:

CompanyName	OrderID
A	10248
B	10249
Null	10250

• FULL [OUTER] JOIN

A FULL OUTER JOIN retrieves records from both tables, including matches and NULL values for non-matching columns.

Syntax:
SELECT column1, column2, ...
FROM Table 1
FULL JOIN Table2 **ON** Table1.Pk = Table2.FK



Orders	
OrderID	CustomerID
10248	1
10249	2
10250	4

Customers	
CustomerID	CompanyName
1	A
2	B
3	C

```
SELECT C.CompanyName, O.OrderID
FROM Customers C FULL JOIN Orders O ON
C.CustomerID = O.CustomerID
```

Result:

CompanyName	OrderID
A	10248
B	10249
C	Null
Null	10250

• CROSS JOIN:

A CROSS JOIN combines each row from one table with every row from another table, producing a Cartesian product of the two tables.

Syntax:
SELECT column1, column2, ...
FROM Table 1
CROSS JOIN Table2

Orders	
OrderID	CustomerID
10248	1
10249	2
10250	4

Customers	
CustomerID	CompanyName
1	A
2	B
3	C

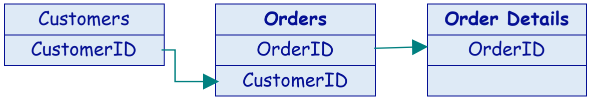
```
SELECT C.CompanyName, O.OrderID
FROM Customers C CROSS JOIN Orders O
```

Result:

CompanyName	OrderID
A	10248
A	10249
A	10250
B	10248
B	10249
B	10250
C	10248
C	10249
C	10250

Querying Multiple Tables:

```
SELECT C.CompanyName, O.OrderID,
P.ProductName
FROM Customers C
       INNER JOIN Orders O
       ON C.CustomerID = O.CustomerID
       INNER JOIN [Order Details] OD
       ON O.OrderID = OD.OrderID
       INNER JOIN Products P
       ON OD.ProductID = P.ProductID
```



Customers
CustomerID

Orders
OrderID
CustomerID

Order Details
OrderID

Subquery:

A subquery, also called an inner select, is a query nested within another query and enclosed in parentheses. It is used to retrieve data from tables based on specified conditions,

Scalar Subquery:

A scalar subquery is a subquery that selects a single column or expression and returns a single value. It can be utilized in any part of an SQL query where a column or expression is used and can be nested within the SELECT, WHERE, or HAVING clause of an outer SELECT statement.

EX:

```
SELECT CompanyName,
      (SELECT COUNT(OrderID)
       FROM Orders O
       WHERE O.CustomerID = C.CustomerID)
       AS OrderCount
FROM Customers C
```

EX:

```
Select *
From Products
Where Unitprice=
(Select MAX(UnitPrice) From Products)
```

SQL Structured Query Language (Northwind DB)

Fatemeh Heydari

EX:

```
SELECT CategoryID, AVG(1.0*UnitPrice) AS
AveragePrice
FROM Products
GROUP BY CategoryID
HAVING AVG(1.0*UnitPrice) >
   (SELECT AVG(1.0*UnitPrice) FROM Products)
```

Multi-Valued Subquery:

A multi-valued subquery returns one or more rows to the outer select statement. It can be used with operators like IN, ANY, SOME, ALL, and EXISTS in the outer query to handle the subquery's multiple rows.

- **ALL**

Syntax:

SELECT column1, column2, ...
FROM Table1
WHERE exp ***Comparison Operator***
   All (**SELECT** exp **FROM** Table2
      WHERE condition)

EX: Retrieve the Customers who don't have orders in the year 1996.

```
SELECT C.CustomerID, C.CompanyName
FROM Customers C
WHERE '1996'<> ALL(SELECT DISTINCT
       CONVERT(VARCHAR(4),
       YEAR(O.OrderDate))
       FROM Orders O
       WHERE O.CustomerID = C.CustomerID
       )
```

- **ANY (or SOME)**

Syntax:

SELECT column1, column2, ...
FROM Table1
WHERE exp ***Comparison Operator***
   ANY (**SELECT** exp **FROM** Table2
      WHERE condition)

EX: Retrieve Customers who have at least one order in the year 1996.

```
SELECT c.CustomerID, c.CompanyName
FROM Customers c
WHERE '1996' = ANY (SELECT DISTINCT
       CONVERT(VARCHAR(4),
       YEAR(o.OrderDate))
       FROM Orders o
       WHERE o.CustomerID = c.CustomerID)
```

- **IN (or EXISTS)**

Syntax:

SELECT column1, column2, ...
FROM Table1
WHERE exp **IN** (**SELECT** exp **FROM** Table2
      WHERE condition)

EX: Retrieve Customers who have never placed an order.

```
SELECT CustomerID, CompanyName
FROM Customers
Where CustomerID NOT IN (SELECT
CustomerID FROM Orders)
```

Ranking Functions:

- **ROW_NUMBER:**

Syntax:

ROW_NUMBER ( ) OVER ([<Partition_by_clause>] < Order_by_clause >)

- **RANK:**

Syntax:

RANK ( ) OVER ([partition_by_clause] order_by_clause)

- **DENSE_RANK:**

Syntax:

DENSE_RANK ( ) OVER ([<Partition_by_clause>] < Order_by_clause >)

- **NTILE:**

Syntax:

NTILE (int_exp) **OVER** ([<partition_by_clause>] < order_by_clause >)

EX:

```
Select ProductName, Unitprice, Rank()
Over(Order BY UnitPrice ASC) AS PriceRank
From Products
```

EX: The following example uses the NTILE function to divide a set of salespersons into four groups based on their Total Sales amount.

```
SELECT E.LastName, NTILE(4) OVER(ORDER BY
SUM(Quantity*Unitprice) DESC) AS Quartile,
CONVERT(VARCHAR(13),
  SUM(Quantity*Unitprice),1) AS TotalSales
FROM Employees AS E
        INNER JOIN Orders AS O
        ON E.EmployeeID = O.EmployeeID
        INNER JOIN [Order Details] AS OD
        ON O.OrderID = OD.OrderID
GROUP BY E.LastName
ORDER BY Quartile, E.LastName
```

Set Operators:

- **UNION:**

  Concatenates the results of two queries into a single result set.

  - **UNION ALL:** Includes duplicates rows.
  - **UNION:** Excludes duplicates rows.

- **EXCEPT:**

  Return distinct rows from the left input query that aren't output by the right input query.

www.linkedin.com/in/fatemeh-heydari-49a2b3238

- **INTERSECT:**

  Return distinct rows that are output by both the left and right input queries operator.

❖ **Conditions For UNION, EXCEPT and INTERSECT:**

1- The number and the order of the columns must be the same in all queries.

2- The data types must be compatible.

EX:

```
SELECT Country FROM Customers --(91 Rows)
UNION ALL
SELECT Country FROM Suppliers --(29 Rows)
```

Result: 120 Rows

EX:

```
SELECT Country FROM Customers --(91 Rows)
EXCEPT
SELECT Country FROM Suppliers --(29 Rows)
```

Result: 9 Rows

EX:

```
SELECT Country FROM Customers --(91 Rows)
INTERSECT
SELECT Country FROM Suppliers --(29 Rows)
```

Result: 12 Rows

SELECT - INTO Clause:

It creates a new table in the default filegroup and inserts the resulting rows from the query into it.

```
SELECT ProductID, ProductName, UnitPrice
INTO P_T
FROM Products
```

INSERT - INTO Clause:

Adds one or more rows to a table or a view in SQL Server.

INSERT INTO Table_Name (column1, column2, ...)
VALUES (value1, value2, ...)

```
INSERT INTO Employees (FirstName, LastName)
Values ('A', 'B')
```

UPDATE:

The "UPDATE" statement is used to modify existing data in a table in a database.

Syntax:

UPDATE <Table Name>|<View Name>
SET <Column1 = Exp1>, <Column2 = Exp2>, ...
[**WHERE** <Condition>]

EX: Increase 2% of the unit price for products, that have a unit price greater than 10.

```
UPDATE Products
SET UnitPrice = UnitPrice * 1.02
WHERE UnitPrice>10
```

DELETE:

Removes one or more rows from a table or view in SQL Server.

Syntax:

DELETE [FROM] <Table Name>|<View Name>
[**WHERE** <Condition>]

EX:

```
DELETE FROM Employees
WHERE LastName='B'
```

Drop Table (Or View):

The DROP TABLE (or View) statement is used to delete a table (Or View) and all its associated data and objects from a database.

Syntax:

DROP TABLE [IF EXISTS] Table_Name [, Table_Name2, ...]

EX:

```
DERP TABLE IF EXISTS P_T
SELECT ProductID, ProductName, UnitPrice
INTO P_T
FROM Products
```

Truncate Table:

Remove all data from the table while keeping the table structure intact.

Syntax:

TRUNCATE TABLE Table_Name

❖ TRUNCATE TABLE cannot truncate tables that are referenced by a FOREIGN KEY constraint. However, you can truncate a table that has a foreign key that references itself.

EX:

```
TRUNCATE TABLE [Order Details]
```

OUTPUT Clause:

Returns information from, or expressions based on, each row affected by an INSERT, UPDATE, DELETE, or MERGE statement.

EX:

```
UPDATE Products
SET UnitPrice= UnitPrice*1.102
OUTPUT deleted.ProductID,
       deleted.ProductName,
       deleted.UnitPrice AS BeforeUpdate,
       inserted.UnitPrice AS AfterUpdate
```