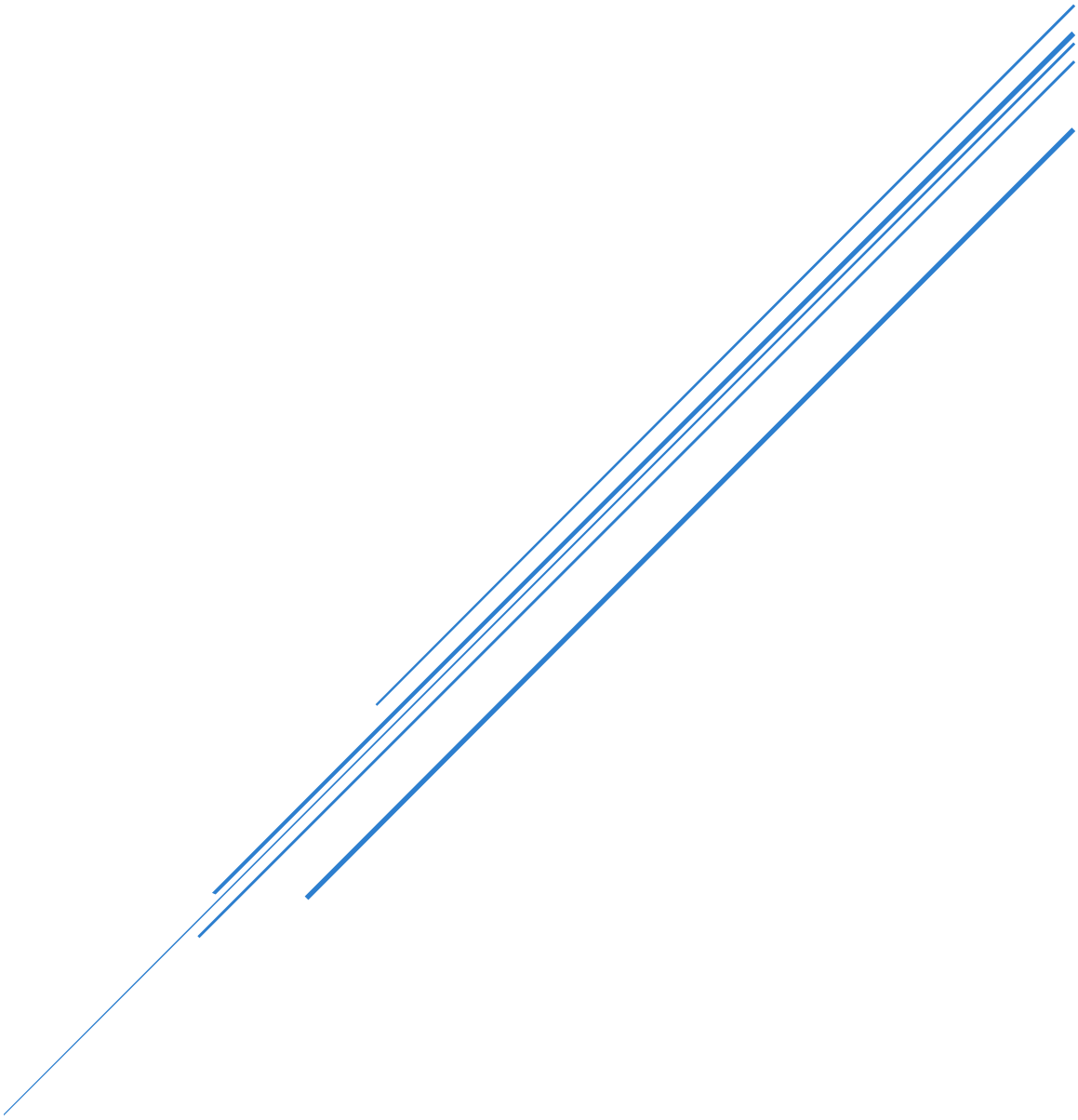
SQL

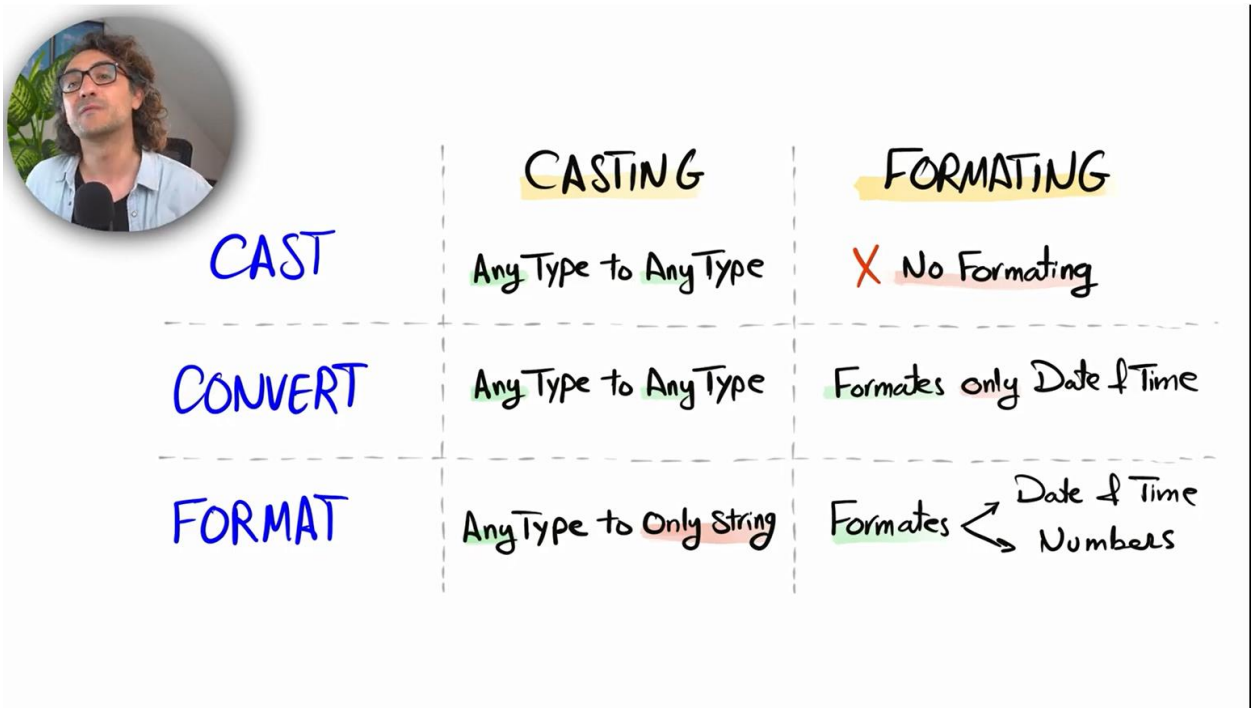
DataWithBaraa



SQL DataWithBaraa

```sql
/*
	CONVERT(): converts a data or time value to a different data type & formats
the value
	------------
	-- syntax --
	------------
	CONVERT(data_type, value, style[optional])
	style in CONVERT() controls how date/time or some numbers are formatted when
converted to text
	style is a number code that tells SQL server which data formats to use when
converting date/time to string

	-- data_type: usually VARCHAR or CHAR
	-- value: date or time
	-- style: format code
*/

USE SalesDB;

SELECT
	CONVERT(VARCHAR, '123') AS [String to Int CONVERT],
	CONVERT(DATE, '2025-08-20') AS [DateTime to Date CONVERT],
	CreationTime,
	CONVERT(DATE, CreationTime) AS [DateTime To Date CONVERT],
	CONVERT(VARCHAR, CreationTime, 32) AS [USA Std. Style: 32],
	CONVERT(VARCHAR, CreationTime, 34) AS [EURO Std. Style: 34]
FROM Sales.Orders;
```

```sql
/*
	========================
	== 34 CAST() Function ==
	========================
	- CAST(): converts a value to a specified data type

	------------
	-- syntax --
	------------
	- CAST(value AS data_type)
*/

USE SalesDB;

SELECT
	CAST('123' AS INT) AS [String To Integer],
	CAST(123 AS VARCHAR) AS [Integer To VARCHAR],
	CAST('2025-08-20' AS DATE) AS [String To Date],
	CAST('2025-08-20' AS DATETIME) AS [String To DateTime],
	CAST('2025-08-20' AS DATETIME2) AS [String To DateTime2],
	CAST(CreationTime AS DATE) AS [Datetime to DATE],
	CAST(CreationTime AS VARCHAR) AS [Datetime to Varchar]
FROM Sales.Orders;
```



Difference Between FORMAT, CONVERT, CAST

Feature	CAST	CONVERT	FORMAT
SQL Standard	✅ Yes (ANSI standard)	❌ No (SQL Server specific)	❌ No (SQL Server specific)
Main Purpose	Convert data type	Convert data type with style	Format output as string
Syntax	CAST(expr AS datatype)	CONVERT(datatype, expr, style)	FORMAT(expr, format, culture)
Style / Format Support	❌ No	✅ Yes (style codes like 101, 103...)	✅ Yes (custom formats)
Output Data Type	Target data type	Target data type	**Always NVARCHAR**
Performance	🚀 Fast	🚀 Fast	🐌 Slow (uses .NET CLR)

Best For	Simple conversions	Date/number formatting in SQL Server	Display/UI formatting
Culture Support	❌ No	❌ No	✅ Yes (en-US, ar-EG, etc.)
Works With	Dates, numbers, strings	Dates, numbers, strings	Dates & numbers only
Recommended in WHERE/JOIN	✅ Yes	✅ Yes	❌ No (bad performance)

```sql
-- CAST
SELECT CAST(GETDATE() AS DATE);

-- CONVERT
SELECT CONVERT(VARCHAR, GETDATE(), 103);
```

```sql
/*
	=============
	== FORMAT ==
	=============
*/

USE SalesDB;

-- format the date
SELECT
	FORMAT(GETDATE(), 'dd/MM/yyyy', 'en-GB') AS [Date];


-- format numbers -> add comma
SELECT
	FORMAT(1234567, 'N') AS [Number];


-- decimal places
SELECT
	FORMAT(1234.567, 'N2') AS [Number];


-- Simple date format
SELECT
	FORMAT(GETDATE(), 'yyyy-MM-dd') AS [Date];


-- custom format
SELECT
	FORMAT(GETDATE(), 'dd/MM/yyyy') AS [Date];
```

```sql
-- get the month name
SELECT
	FORMAT(GETDATE(), 'MMMM') AS [Month];
```

```sql
/*
	================
	== 35 DATEADD ==
	================
	- DATEADD(): is a SQL function used to add or subtract a time interval (days,
months, years, etc.) to/from a date.

	------------
	-- syntax --
	------------
	DATEADD(datepart, interval, date)

	Parameters:
		datepart: the unit of time (year, month, day, etc.)
		number: how many units tWo add (use a negative number to subtract)
		date: the original date


	| datepart    | Meaning |
	| ----------- | ------- |
	| year / yy   | Year    |
	| month / mm  | Month   |
	| day / dd    | Day     |
	| hour / hh   | Hour    |
	| minute / mi | Minute  |
	| second / ss | Second  |

*/
USE SalesDB;

-- add 10 days to 2026-01-29
SELECT
'2026-01-29',
DATEADD(day, 10, '2026-01-29');


-- subtract two months from 2026-01-29
SELECT
'2026-01-29',
DATEADD(month, -2, '2026-01-29');


-- add one year to 2026-01-29
SELECT
	'2026-01-29',
	DATEADD(year, 1, '2026-01-29');


-- Date after 30 days from today
SELECT
	GETDATE(),
	DATEADD(day, 30, GETDATE());
```

```sql
-- Calculates the delivery date 7 days after the order date.
USE MyDatabase;

SELECT
    order_date,
    DATEADD(day, 7, order_date) AS DeliveryDate
FROM Orders;


-- add one year to the CreationTime
USE SalesDB;
SELECT
      CreationTime,
      DATEADD(YEAR, 1, CreationTime) AS [Add One Year]
FROM Sales.Orders;
```

```sql
/*
      =================
      == 36 DATEDIFF ==
      =================
      - DATEDIFF() is a SQL function used to calculate the difference between two
dates.
      It returns the difference as an integer, based on the date part you specify
(days, months, years, etc.).

      allow us to find the differences between two dates

      ------------
      -- syntax --
      ------------
      - DATEDIFF(part, start_date, end_date)
*/

USE SalesDB;

SELECT
      DATEDIFF(month, '2025-01-15', '2026-01-15') AS MonthsDifference;

-- how many days
SELECT
      DATEDIFF(DAY, '2025-01-15', '2026-01-15') AS MonthsDifference;


-- calculate the age of employees
SELECT
      BirthDate,
      DATEDIFF(YEAR, BirthDate, GETDATE()) AS [Employee Age]
FROM Sales.Employees

/*
      LAG() is a window (analytic) function in SQL that lets you look at a previous
row's
      value in the same result set without using a self-join.

      LAG(column_name, offset, default_value) OVER (PARTITION BY partition_column
ORDER BY order_column)
```

```sql
*/


/*
	find the average shipping duration in days for each month
	shipping duration = OrderDate - ShipDate
*/
SELECT
	MONTH(OrderDate) AS [Order Date],
	AVG(DATEDIFF(DAY, OrderDate, ShipDate)) AS [Day to Ship]
FROM Sales.Orders
GROUP BY MONTH(OrderDate);


/*
	find the number of days between each order and previous order
	this question called -> 'time gap analysis'
*/
SELECT
	OrderID,
	OrderDate AS [Current Order Date],
	LAG(OrderDate) OVER (ORDER BY OrderDate) AS [Previous Order Date],
	DATEDIFF(DAY, LAG(OrderDate) OVER (ORDER BY OrderDate), OrderDate) AS [Number
of Days]
FROM Sales.Orders;
```

```sql
/*
	============
	== ISDATE ==
	============
	- ISDATE() is a SQL Server function that checks whether a given expression
can be converted to a valid date.
	It’s useful for validating data before doing date operations.

	------------
	-- syntax --
	------------
	ISDATE(expression)

*/

SELECT
    ISDATE('2026-01-31'),
    ISDATE('31-01-2026'),
    ISDATE('hello');
```

```sql
/*
	====================
	== NULL Functions ==
	====================
	- NULL means no value / unknown / missing value.
	- NULL is not equal to anything
	- It does NOT mean: 0, empty string '', space ' ', It literally means “there
is nothing here”.
```

```
	--------------
	-- ISNULL() --
	--------------
	- ISNULL() is a SQL Server function used to replace NULL with a value you
choose.


	----------------
	-- COALESCE() --
	----------------
	- COALESCE() returns the first NON-NULL value from a list of expressions.

	-----------
	-- syntax -
	-----------
	 COALESCE(value1, value2, value3, ...)
*/

USE SalesDB;

SELECT ISNULL(NULL, 0) AS [Result];      -- Result: 0
SELECT ISNULL(NULL, 'N/A') AS [Result];  -- Result: N/A
SELECT ISNULL(NULL, NULL) AS [Result];  -- Result: NULL


SELECT COALESCE(NULL, NULL, 10) AS [Result];   -- Result: 10
SELECT COALESCE(NULL, 'A', 'B') AS [Result];   -- Result: A


SELECT COALESCE(NULL, NULL, 0) AS [Result];       -- INT
SELECT COALESCE(NULL, NULL, 'text') AS [Result];  -- VARCHAR


-- Show customer name and score. If score is NULL, show 0.
SELECT
    first_name,
    ISNULL(score, 0) AS score
FROM customers;


-- Missing country, If country is NULL, show 'Unknown'.
SELECT
    first_name,
    ISNULL(country, 'Unknown') AS country
FROM customers;


-- Fix messy names. Trim the name and replace NULL with 'No Name'.
SELECT
    ISNULL(LTRIM(RTRIM(first_name)), 'No Name') AS first_name,
    country,
    score
FROM customers;
```

```sql
-- Math with ISNULL(). Add 50 bonus points. If score is NULL treat it as 0.
SELECT
    first_name,
    score,
    ISNULL(score, 0) + 50 AS score_with_bonus
FROM customers;


-- Filtering with ISNULL. Show customers with score < 500. (Treat NULL score as 0).
SELECT
    first_name,
    country,
    score
FROM customers
WHERE ISNULL(score, 0) < 500;


-- Grouping + ISNULL (important!). Show average score per country
SELECT
    country,
    AVG(ISNULL(score, 0)) AS avg_score
FROM customers
GROUP BY country;


-- Replace NULL score. If score is NULL, show 0.
SELECT
    first_name,
    COALESCE(score, 0) AS score
FROM customers;


-- Multiple fallbacks (very realistic). If first_name is NULL -> use 'Unknown'.
SELECT
    COALESCE(first_name, 'Unknown') AS first_name,
    country,
    score
FROM customers;


-- Clean names + COALESCE. Some names have spaces (' John', ' hassan ').
SELECT
    COALESCE(NULLIF(LTRIM(RTRIM(first_name)), ''), 'No Name') AS clean_name,
    country,
    score
FROM customers;


-- COALESCE in math
SELECT
    first_name,
    score,
    COALESCE(score, 0) + 100 AS bonus_score
FROM customers;
```

```sql
-- Filtering with COALESCE. Show customers with score < 500 (NULL = 0).
SELECT
    first_name,
    country,
    score
FROM customers
WHERE COALESCE(score, 0) < 500;


-- Grouping + COALESCE. Average score per country (NULL = 0).
SELECT
    country,
    AVG(COALESCE(score, 0)) AS avg_score
FROM customers
GROUP BY country;

/*
    Why COALESCE is better here
        - Works in all databases
        - Handles multiple options
        - Safer data types than ISNULL
*/


-- find the average scores of the customers
USE MyDatabase;

SELECT
    id AS [customer id],
    score,
    COALESCE(score, 0) AS [score 2],
    AVG(score) OVER() AS [Average Score],
    AVG(COALESCE(score, 0)) OVER() AS [Average Score 2]
FROM customers;
-- handle the NULL befoe doing any mathematical operations
SELECT
    5 + NULL, -- Result = NULL
    5 + 5, -- Result = 10
    'A' + 'B', -- Result = AB
    'A' + NULL -- Result = NULL

/*
    display the full name of customers in a single filed by merging their first and
last name
    and add 10 bonus points to each customer's score
*/
USE SalesDB;

SELECT
    CustomerID,
    FirstName,
    LastName,
    FirstName + ' ' + COALESCE(LastName, '') AS [Full Name],
    score as [score],
    COALESCE(score, 0) + 10 AS [score with bonus]
FROM sales.Customers;
```

```
/*
    ------------------------------------------
    -- handle the NULL before doing any JOINS --
    ------------------------------------------
    if there are any NULL values inside any table, will loss the record at the
output
    therefore must handle the NULL before doing any JOINS
*/
```

Feature	ISNULL()	COALESCE()
Type	SQL Server function	ANSI SQL standard
Portability	❌ SQL Server only	✅ Works in most DBs
Number of arguments	2 only	2 or more
Returns first non-NULL	❌ (only checks 1 value)	✅
Data type handling	Returns type of **1st argument**	Uses **data type precedence**
Allows all NULL constants	✅	❌ (needs typed expression)
Performance	Slightly faster (SQL Server)	Slightly slower (negligible)

How to get the data type of each column on the table

```
USE SalesDB;

SELECT
    COLUMN_NAME,
    DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'Sales'
AND TABLE_NAME = 'Customers';
```

```sql
/*
	====================
	== CASE Statement ==
	====================
	- It lets you add conditions inside a query and return different values based
on those conditions.
    - evaluates a list of conditions and returns a value when the first condition is
met
    - the main purpose of CASE statement is "data transformation"
    - drive new information [create new columns based on existing data]
    - CATEGORIZING THE DATA -> group the data into different categorizes based on
certain conditions


	------------
	-- syntax --
	------------
	CASE
		WHEN condition1 THEN result1
		WHEN condition2 THEN result2
		WHEN condition3 THEN result3
		ELSE default_result


*/

USE MyDatabase;


-- Categorize Customers by Score
SELECT
    first_name,
    score,
    CASE
        WHEN score >= 80 THEN 'High Score'
        WHEN score >= 50 THEN 'Medium Score'
        ELSE 'Low Score'
    END AS score_category
FROM customers;


-- Label Local vs Foreign Customers
SELECT
    first_name,
    country,
    CASE
        WHEN country = 'Egypt' THEN 'Local'
        ELSE 'Foreign'
    END AS customer_type
FROM customers;
```

```sql
-- Replace NULL Scores
SELECT
    first_name,
    CASE
        WHEN score IS NULL THEN 0
        ELSE score
    END AS cleaned_score
FROM customers;


-- Create Customer Rank Levels
SELECT
    first_name,
    score,
    CASE
        WHEN score >= 90 THEN 'VIP'
        WHEN score >= 70 THEN 'Gold'
        WHEN score >= 50 THEN 'Silver'
        ELSE 'Bronze'
    END AS customer_rank
FROM customers;


-- Conditional Aggregation - Count customers by score level:
SELECT
    COUNT(CASE WHEN score >= 80 THEN 1 END) AS high_score_customers,
    COUNT(CASE WHEN score BETWEEN 50 AND 79 THEN 1 END) AS medium_score_customers,
    COUNT(CASE WHEN score < 50 THEN 1 END) AS low_score_customers
FROM customers;



-- Create Pass/Fail Flag
SELECT
    first_name,
    score,
    CASE
        WHEN score >= 50 THEN 'Passed'
        ELSE 'Failed'
    END AS result
FROM customers;


-- Handle Multiple Countries with Labels
SELECT
    first_name,
    country,
    CASE
        WHEN country IN ('Egypt', 'Saudi Arabia', 'UAE') THEN 'Middle East'
        WHEN country IN ('USA', 'Canada') THEN 'North America'
        ELSE 'Other Region'
    END AS region
FROM customers;

-----------------------------------------------------------------------
```

```sql
USE SalesDB;

/*
    generate a report showing the total sales for each category
        - High: if the sales higher than 50
        - Medium: if the sales between 20 and 50
        - Low: if the sales equal or lower than 20
    and sort the result from lowest to highest
*/
SELECT
    Category,
    SUM(sales) AS [total sales]
FROM (
    SELECT
        OrderID,
        Sales,
    CASE
        WHEN Sales > 50 THEN 'High'
        WHEN Sales BETWEEN 20 AND 50 THEN 'Medium'
        WHEN Sales <= 20 THEN 'Low'
    END 'Category'
    FROM Sales.Orders
)t
GROUP BY Category
ORDER BY [total sales] ASC


-----------------------------
-- Rules of CASE statement --
-----------------------------
-- the data type of the results must be matching, means the results must be string,
numbers, dates, ...
-- CASE statement can be used anywhere in the

/*
    ====================
    == Mapping Values ==
    ====================
    - transform the values from one form to another in order to make it more
readable and re-usable for analytics
*/

-- retrieve employee details with gender displayed as full text
SELECT
    EmployeeID,
    FirstName,
    LastName,
    Gender,
CASE
    WHEN Gender = 'M' THEN 'Male'
    WHEN Gender = 'F' THEN 'Female'

    -- if there are NULL values
    ELSE 'Not Available'
END [Gender Info]
FROM Sales.Employees;
```

```sql
-- retrieve customer details with abbreviated country code
SELECT
    CUstomerID,
    FirstName,
    LastName,
    Country,
CASE
    WHEN Country = 'Germany' THEN 'DE'
    WHEN Country = 'USA' THEN 'US'
    ELSE 'n/a'
END 'Country Abbreviation'
FROM Sales.Customers;



/*
    ==================
    == Handle NULLs ==
    ==================
    - replace NULLs with a specific values
    - NULLs can lead to inaccurate results
    - which can lead to wrong decision-making
*/


-- find the average scores of customers and treat NULLs as 0
SELECT
    CustomerID,
    FirstName,
    LastName,
    Score,
CASE
    WHEN Score IS NULL THEN 0
    ELSE Score
END [Clean Score],
AVG(Score) OVER () AS [AverageScore],
AVG(CASE
        WHEN Score IS NULL THEN 0
        ELSE Score
    END) OVER() AvgCustomerClean
FROM Sales.Customers;


/*
    ============================
    == Conditional Aggregation ==
    ============================
    - apply aggrgate functions only on subsets of data that fulfill certain
conditions
*/
```

```sql
-- count how many times each customer has made an order with sales greater than 30
SELECT
    CustomerID,
    SUM(CASE
        WHEN Sales > 30 THEN 1
        ELSE 0
    END) AS TotalOrdersHighSales,
    COUNT(*) AS TotalOrders
FROM Sales.Orders
GROUP BY CustomerID
ORDER BY CustomerID;
```